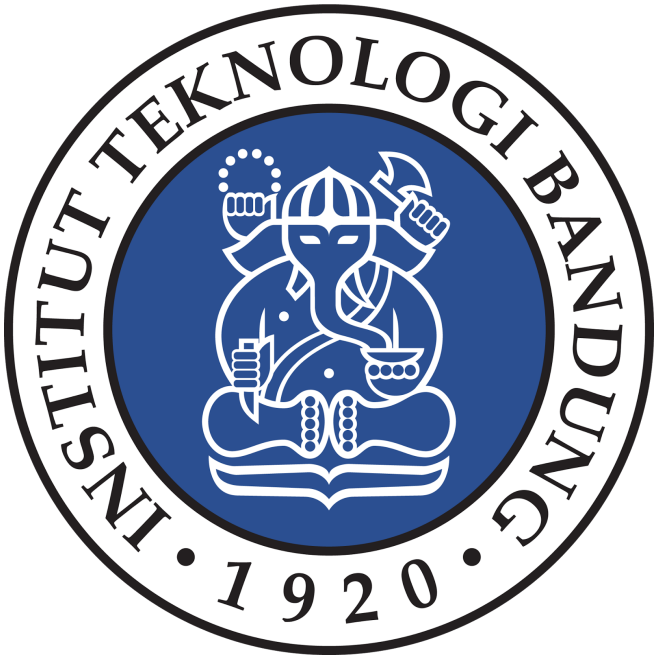
LAPORAN
TUGAS KECIL 2 IF2211
STRATEGI ALGORITMA

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



oleh

Go Dillon Audris 13521062
Austin Gabriel Pardosi 13521084

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2022 / 2023

DAFTAR ISI

DAFTAR ISI **2**

BAB I **4**

PENJELASAN ALGORITMA PROGRAM **4**

1.1 Deskripsi Permasalahan Titik Terdekat dan Pencarian Solusi **4**

1.2 Implementasi Algoritma *Divide and Conquer* untuk Mencari Solusi **5**

1.2.1 Tahap Praproses Pencarian Solusi 5

1.2.2 Tahap Pencarian Solusi 6

1.2.3 Tahap Pemrosesan Tambahan 7

1.3 Analisis Kompleksitas Waktu Algoritma *Divide and Conquer* 8

1.4 Implementasi Bonus pada Program 9

1.4.1 Penggambaran Titik pada Ruang 3 Dimensi 9

1.4.2 Generalisasi Program untuk Titik pada Ruang R^n 9

BAB II **10**

SOURCE CODE PROGRAM ***10***

2.1 *File Dot.py* 10

2.2 *File ListDot.py* 11

2.3 *File IO.py* 13

2.4 *File Main.py* 14

BAB III **17**

EKSPERIMEN DAN *TESTING* **17**

3.1 Testing untuk Titik pada Ruang R^3 17

3.1.1 Banyak Titik = 16 17

3.1.2 Banyak Titik = 64 18

3.1.3 Banyak Titik = 128 19

3.1.4 Banyak Titik = 1000 20

3.2 Testing untuk Titik pada Ruang R^n dengan $n \leq 2$ 21

3.2.1 Banyak Titik = 16 21

3.2.2 Banyak Titik = 64 22

3.2.3 Banyak Titik = 128 23

3.2.4 Banyak Titik = 1000 24

3.3 Testing untuk Titik pada Ruang R^n dengan $n > 3$ 25

3.3.1 Banyak Titik = 16 25

3.3.2 Banyak Titik = 64 26
3.3.3 Banyak Titik = 128 27
3.3.4 Banyak Titik = 1000 28

DAFTAR PUSTAKA **29**

LAMPIRAN **30**
Link Repository Program *31*
Checklist Tugas Kecil *31*

BAB I

PENJELASAN ALGORITMA PROGRAM

1.1 Deskripsi Permasalahan Titik Terdekat dan Pencarian Solusi

Dalam permasalahan pencarian pasangan titik terdekat dalam ruang R^3, diberikan suatu himpunan titik (P) yang terdiri atas N buah titik yang dapat direpresentasikan dengan vektor (x_i, y_i, z_i, i = 1, 2, .. , N). Solusi dari permasalahan ini adalah pasangan titik yang memiliki jarak paling pendek. Jarak tersebut dapat dihitung dengan menggunakan rumus Euclidean berikut:

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

Algoritma *Divide and Conquer* dapat digunakan untuk menemukan solusi dari permasalahan ini. Dalam ruang R^2, terdapat suatu garis vertikal maya L yang membagi himpunan titik (P) menjadi dua upa-himpunan dengan banyak titik yang kurang lebih sama. Garis L dihampiri dengan memanfaatkan L = nilai x dari titik yang berada di tengah senarai titik yang telah diurutkan. Pada ruang R^3, garis vertikal maya L dengan diganti dengan suatu bidang D yang sejajar dengan bidang *yz* dan terletak pada x = nilai x dari titik yang berada di tengah senarai titik yang telah diurutkan. Bidang D akan membagi himpunan titik (P) menjadi dua upa-himpunan dengan ukuran yang kurang lebih sama. Dengan pembagian himpunan ini, maka algoritma *divide and conquer* dapat digunakan.

Dalam implementasinya, algoritma bekerja dengan membagi dua himpunan titik (P) menjadi upa-himpunan yang mengandung jumlah titik yang lebih sedikit dari N. Pasangan titik terdekat lalu dicari pada kedua upa-himpunan dengan memanfaatkan kembali algoritma *divide and conquer*. Jika jumlah titik pada upa-himpunan sudah cukup kecil, maka pasangan titik

terdekat dapat ditemukan dengan memanfaatkan algoritma *bruteforce*. Selain pada kedua upa-himpunan, pasangan titik terdekat juga dicari pada titik-titik yang tidak berada pada himpunan yang sama. Ketiga pasangan titik terdekat lalu dibandingkan untuk mendapatkan pasangan titik terdekat absolut pada himpunan P. Cara kerja algoritma ini juga dapat digunakan untuk mendapatkan pasangan titik terdekat pada ruang R^n.

1.2 Implementasi Algoritma *Divide and Conquer* untuk Mencari Solusi

1.2.1 Tahap Praproses Pencarian Solusi

Diasumsikan bahwa program telah menerima dua buah masukan berupa banyak titik yang ada (N) dan ruang dimensi dari titik tersebut (n). Program lalu akan menciptakan suatu senarai berukuran N yang elemennya adalah objek titik. Objek titik memiliki suatu senarai berukuran n untuk merepresentasikan vektor titik tersebut. Nilai dari elemen vektor dibangkitkan secara acak dalam rentang -10000 sampai 10000.

Sebelum algoritma *divide and conquer* dapat dilakukan, senarai titik harus diurutkan terlebih dahulu. Hal ini supaya algoritma *divide and conquer* menjadi lebih efisien dalam mempartisi senarai menjadi dua upa-senarai. Pengurutan dilakukan berdasarkan nilai elemen pertama dari vektor titik (x) dan diimplementasikan menggunakan metode *quicksort* dengan pivot berupa titik yang berada di tengah senarai. Pada akhir dari setiap pemrosesan partisi senarai, pivot akan berada pada urutan yang benar. Semua titik dengan nilai x yang lebih kecil akan berada di sebelah kiri pivot, dan semua titik dengan nilai x yang lebih besar akan berada di sebelah kanan pivot. *Quicksort* kemudian akan dipanggil lagi untuk mengurutkan kedua upa-senarai di sebelah kiri dan kanan pivot. Hasil dari runtutan pemanggilan *quicksort* adalah senarai titik dengan nilai elemen pertama vektor titik (x) terurut membesar.

1.2.2 Tahap Pencarian Solusi

Setelah senarai diurutkan, maka algoritma *divide and conquer* akan dijalankan terhadap senarai tersebut. Algoritma *divide and conquer* untuk mendapatkan pasangan titik terdekat dapat dipecah menjadi bagian-bagian berikut:

1). Jika banyak titik pada senarai atau upa-senarai sudah cukup sedikit (≤ 3), maka implementasikan algoritma *bruteforce* untuk mencari pasangan titik terdekat.

- Algoritma *bruteforce* dilakukan dengan membandingkan jarak euclidean setiap titik pada senarai atau upa-senarai dengan titik lainnya. Algoritma *bruteforce* dioptimasi dengan pendekatan heuristik, sehingga perbandingan jarak hanya dilakukan pada pasangan titik yang belum dihitung jaraknya. Hal ini dilakukan karena jarak euclidean antara titik a dengan titik b sama dengan jarak euclidean antara titik b dengan titik a.

2). Jika banyak titik pada senarai masih cukup besar (> 3), maka bagilah senarai menjadi dua upa-senarai dengan ukuran yang kurang lebih sama. Pembagian dilakukan berdasarkan nilai elemen pertama vektor titik (x) yang berada di tengah senarai. Hasilnya adalah dua upa-senarai dengan upa-senarai yang berada di "kiri" titik tengah memiliki nilai x $\leq$ nilai x titik tengah, dan upa-senarai yang berada di "kanan" titik tengah memiliki nilai x $\geq$ nilai x titik tengah. Perhatikan bahwa pembagian senarai tetap mempertahankan keterurutan objek-objek titik pada upa-senarainya.

3). Lakukan langkah 2 secara rekursif pada setiap upa-senarai, dan lakukan langkah 1 jika upa-senarai sudah cukup kecil.

4). Solusi permasalahan ini memiliki 3 kemungkinan :

a. Pasangan titik terdekat berasal dari upa-senarai kiri

b. Pasangan titik terdekat berasal dari upa-senarai kanan

c. Pasangan titik terdekat adalah 2 titik yang berasal dari upa-senarai yang berbeda.

Solusi a dan b telah ditemukan lewat langkah 3. Namun, solusi c memerlukan pemrosesan tambahan.

1.2.3 Tahap Pemrosesan Tambahan

Telah dijelaskan pada langkah 2 pada bagian subbab 1.2.2 bahwa senarai dibagi berdasarkan nilai elemen pertama vektor titik (x) yang berada di tengah senarai. Nilai x inilah yang menjadi tempat bidang D yang sejajar dengan bidang *yz* berada dan membagi senarai menjadi dua upa-senarai. Untuk menemukan solusi c dari langkah 4 pada subbab 1.2.2, perlu diketahui terlebih dahulu suatu nilai δ (delta). Nilai δ didapatkan dengan mencari jarak terkecil antara solusi a dan solusi b.

Nilai δ akan digunakan untuk memperlebar bidang D menjadi balok dengan lebar ((x+δ)-(x-δ)) dan panjang serta tinggi yang tak terhingga. Selanjutnya, akan ditemukan seluruh titik pada senarai yang berada di dalam balok D tersebut. Titik-titik inilah yang menjadi kandidat pasangan titik terdekat untuk solusi c. Kita tidak perlu mengecek seluruh titik dari kedua upa-himpunan karena jarak dari komponen x kedua titik lebih besar dari nilai δ yang mengindikasikan bahwa jaraknya lebih besar dari jarak terpendek yang telah ditemukan.

Selanjutnya, untuk seluruh titik yang berada di dalam bidang D, akan dilakukan pencarian pasangan titik terdekat dengan algoritma yang mirip seperti algoritma *bruteforce* pada langkah 1. Perbedaan algoritma ini adalah tambahan metode heuristik sebelum jarak euclidean kedua titik dihitung. Dilakukan pengecekan terlebih dahulu terhadap jarak dari komponen-komponen vektor kedua titik. Jika ada satu saja jarak komponen vektor kedua titik yang lebih besar dari nilai δ, maka perhitungan jarak euclidean tidak perlu dilakukan karena jarak euclideannya sudah pasti akan lebih besar dari δ.

Hasil dari pemrosesan ini adalah pasangan titik terdekat yang berasal dari dua upa-senarai yang berbeda. Pasangan titik ini menjadi solusi c. Langkah terakhir dari algoritma *divide and conquer* adalah membandingkan solusi a, b, dan c. Perbandingan ketiga solusi akan mendapatkan pasangan titik terdekat absolut untuk himpunan titik (P).

1.3 Analisis Kompleksitas Waktu Algoritma *Divide and Conquer*

Dengan penjelasan algoritma *divide and conquer* pada subbab 1.2, kompleksitas waktu algoritma ini dapat dihitung. Kompleksitas waktu dari tahap praproses berupa pengurutan senarai dengan metode *quicksort* dapat diabaikan karena tidak termasuk ke dalam algoritma *divide and conquer*.

Algoritma *divide and conquer* yang diimplementasikan membagi persoalan besar dengan ukuran N titik menjadi persoalan yang lebih kecil dengan ukuran kurang lebih N/2 titik. Persoalan yang lebih kecil ini kemudian diselesaikan lagi dengan memanfaatkan algoritma *divide and conquer*. Selain membagi persoalan menjadi 2, ada pemrosesan tambahan untuk memeriksa pasangan titik terdekat yang berasal dari dua upa-senarai yang berbeda. Hal ini menyebabkan penambahan kompleksitas algoritma sebesar cN, dengan c adalah suatu konstanta. Oleh karena itu, kompleksitas dari algoritma *divide and conquer* dapat dirumuskan sebagai berikut:

$$T(n) = 2.T(n/2) + cn, \quad n > 3$$

$$T(n) = a \qquad\qquad , n \leq 3$$

Terlihat bahwa T(n) untuk n > 3 dapat diselesaikan menggunakan Teorema Master dengan a = 2, b = 2, dan d = 1. Terdapat hubungan $a = b^d$, sehingga notasi Big-O nya adalah $O(n^d \log n) = O(n \log n)$

1.4 Implementasi Bonus pada Program

1.4.1 Penggambaran Titik pada Ruang 3 Dimensi

Penggambaran titik pada ruang 3 dimensi menggunakan *library matplotlib.pyplot*. Implementasi penggambaran titik ini dimulai dengan membuat *figure* dan *axes* 3D pada sebuah plot. Langkah selanjutnya, *method* ini akan membuat beberapa titik secara 3 dimensi dengan kalang untuk setiap *scatter*. Semua titik yang dihasilkan akan berwarna biru dan berbentuk ‘o’ kecuali untuk sepasang titik yang terdekat yang akan berwarna merah dan berbentuk ‘x’. Terakhir *method* akan memberikan label di masing-masing sumbu dan akan menampilkan visualisasinya secara 3 dimensi.

1.4.2 Generalisasi Program untuk Titik pada Ruang R^n

Generalisasi program untuk mencari pasangan titik terdekat pada dimensi yang lebih tinggi dilakukan dengan menambahkan permintaan input pada pengguna sebagai masukan untuk banyak dimensi dari titik yang akan dibangkitkan. Implementasi algoritma *divide and conquer* program masih menggunakan konsep yang sama yang dijelaskan pada subbab 1.2, namun dengan pengecekan tambahan. Tambahan pengecekan dilakukan pada pemrosesan yang dijelaskan pada subbab 1.2.3. Algoritma tidak lagi hanya mengecek jarak dari komponen y dan z kedua titik, namun mengecek seluruh jarak komponen kedua titik hingga komponen ke n (n adalah jumlah dimensi titik).

BAB II

SOURCE CODE PROGRAM

2.1 *File Dot.py*

```python
import random
import math

class Dot:
    def __init__(self, dimension):
        # Initiate the number of dimension of dot and its position
        self.dimension = dimension
        self.position = [round(random.uniform(-10000, 10000), 3) for i in range
(dimension)]
    def __sub__(self, other):
        # EUCLIDEAN DISTANCE FUNCTION
        # Calculate the distance between this dot and other dot
        distance_squared = 0

        for i in range(0, self.dimension):
            distance_squared += math.pow((other.position[i]-self.position[i]), 2)

        return math.sqrt(distance_squared)

    def checkDimensionDistance(self, other, mindist) :
        check = True
        for i in range(1, self.dimension) :
            if (abs(other.position[i] - self.position[i]) >= mindist) :
                check = False
                break
        return check

    def printDot(self):
        # Print the position of this dot
        print(f"[{self.position[0]}", end="")
        for i in range(1, self.dimension):
            print(f", {self.position[i]}", end="")
        print("]")

    def writeDot(self):
        # Return string containing the dot coordinate information
        coor = f"[{self.position[0]}"
        for i in range(1, self.dimension):
            coor += f", {self.position[i]}"
        coor += "]"
        return coor
```

Gambar 2.1 Source Code File Dot.py

2.2 *File ListDot.py*

```python
from Dot import *
import matplotlib.pyplot as plt
import numpy as np
from ListDot import *

class ListDot:
    # ============================== CONSTRUCTOR OF LIST ==============================
    def __init__(self, dimension, neff):
        # Initiate the number of dots, dimension of dot, list of dots, and number of euclid distance function called
        self.neff = neff
        self.dimension = dimension
        self.euclid_count = 0
        self.buffer = [Dot(self.dimension) for i in range (self.neff)]

        # Sort the array according to x-coordinate with quicksort algorithm
        self.quickSort(0, self.neff-1)

    def quickSort(self, start, finish):
        # Initialize the pointer
        left = start
        right = finish

        # If the array contains more than 1 element, do the quicksort
        if (start < finish):
            i = self.partition(start, finish)
            self.quickSort(start, i-1)
            self.quickSort(i+1, finish)

    def partition(self, start, finish):
        # Initialize the pointer and pivot
        left = start
        right = finish
        pivotVal = self.buffer[(start+finish) // 2].position[0]

        # Partition the array into left and right. End of partition, pivot in the correct place
        while (left <= right):
            while (self.buffer[left].position[0] < pivotVal):
                left += 1

            while (self.buffer[right].position[0] > pivotVal):
                right -= 1

            self.buffer[left], self.buffer[right] = self.buffer[right], self.buffer[left]

        # Undo the last swap
        self.buffer[left], self.buffer[right] = self.buffer[right], self.buffer[left]

        # Return the pivot index
        return right

    # ================================ BRUTE FORCE ================================
    def bruteForce(self, start, finish):
        # Initialize the value
        minDist = np.inf
        dot1 = Dot(self.dimension)
        dot2 = Dot(self.dimension)
        euclid_count = 0

        # Calculate the closest pair of dots using brutefore
        for i in range (start, finish):
            for j in range(i+1, finish+1):
                euclid_count += 1
                temp = self.buffer[j] - self.buffer[i]
                if (temp < minDist):
                    minDist = temp
                    dot1 = self.buffer[i]
                    dot2 = self.buffer[j]

        # Return the closest pair of dots and their distance
        return dot1, dot2, minDist, euclid_count
```

```python
    # ============================== DIVIDE AND CONQUER ==============================
    def divideAndConquer(self, start, finish):
        # Initialize the value
        minDist = 999999
        dot1 = Dot(self.dimension)
        dot2 = Dot(self.dimension)

        # Process the list depending on its size
        if (finish-start+1 <= 3):
            # Find the closest pair of dots using bruteforce
            dot1, dot2, minDist, count = self.bruteForce(start, finish)
            self.euclid_count += count
        else:
            # Define the size of current partition
            size = finish - start + 1

            # Partition the list of dots into left and right
            dot3, dot4, minDistLeft = self.divideAndConquer(start, start + (size // 2) - 1)
            dot5, dot6, minDistRight = self.divideAndConquer(start + (size // 2), finish)

            # Determine the minimum distance (either from left partition or right partition)
            if (minDistLeft <= minDistRight):
                dot1, dot2, minDist = dot3, dot4, minDistLeft
            else:
                dot1, dot2, minDist = dot5, dot6, minDistRight

            # Calculate the closest pair of dots across the midline with width current minDist
            # Find x point where it is the middle point among all dots
            locX = (finish + start) // 2
            midX = self.buffer[locX].position[0]

            # Make an array of all dots from mindist to left and minDist to right
            p = start
            while abs(self.buffer[p].position[0] - midX) > minDist:
                p += 1
            left = p
            p += 1
            while  p <= finish and abs(self.buffer[p].position[0] - midX) <= minDist:
                p += 1
            right = p-1

            # Compare the minDist between pair of dots across the midline and pair of dots from the same partition
            for i in range (left, right) :
                for j in range (i+1, right+1) :
                    if (self.buffer[i].checkDimensionDistance(self.buffer[j], minDist)) :
                        jarak = self.buffer[i]-self.buffer[j]
                        self.euclid_count += 1
                        if jarak < minDist :
                            minDist = jarak
                            dot1, dot2 = self.buffer[i], self.buffer[j]
        return dot1, dot2, minDist
```

```python
    # ============================== HELPER SECTION ================================
    def printAllDots(self):
        # Print the information of all dots. For debugging
        for i in range(self.neff):
            self.buffer[i].printDot()
            print()

    def plotFor3D(self, dot1, dot2):
        # Create a 3D plot
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')

        # Generate some 3D data
        for i in range(self.neff) :
            if (self.buffer[i] != dot1 and self.buffer[i] != dot2) :
                ax.scatter(self.buffer[i].position[0], self.buffer[i].position[1], self.buffer[i].position[2],
c='blue', marker='o')

        # Creating the title
        ax.set_title("Pasangan Titik Terdekat 3D")

        # Plot the nearest dots using scatter plot
        ax.scatter(dot1.position[0], dot1.position[1], dot1.position[2], c='red', marker='x')
        ax.scatter(dot2.position[0], dot2.position[1], dot2.position[2], c='red', marker='x')

        # Set the labels name for the axes
        ax.set_xlabel('Sumbu X')
        ax.set_ylabel('Sumbu Y')
        ax.set_zlabel('Sumbu Z')

        # Show the plot
        plt.show()
```

Gambar 2.2 Source Code File ListDot.py

2.3 *File IO.py*

```python
class IO:
    @staticmethod
    def saveToFile(answer, name):
        with open(f"../test/{name}.txt", "w") as file:
            file.write(answer)
            file.close()
```

Gambar 2.3 Source Code File IO.py

2.4 *File Main.py*

```python
import time
from ListDot import *
from IO import *

class Main:
    @staticmethod

    def askForDotNumber():
        # Validate the user input of number of dot
        while True:
            try:
                neff = int(input("Please input the number of dots : "))
                if (neff < 2):
                    raise ValueError
            except ValueError:
                print("Please input an integer (>= 2)")
                continue
            else:
                return neff
                break

    def askForDimension():
        # Validate the user input of number of dimension of the dot
        while True:
            try:
                dimension = int(input("Please input the number of dimension of the dot : "))
            except ValueError:
                print("Please input an integer")
                continue
            else:
                return dimension
                break

    def askForSaveToFile():
        # Validate the user input of option to save the answer to file or not
        while True:
            try:
                opt = input("Do you want to save your answer (Y/n): ").lower()
                if (opt != "y" and opt != "n"):
                    raise ValueError
            except ValueError:
                print("Please answer with yes (Y) or no (n)")
                continue
            else:
                return opt
                break

    def askForVisualizer():
        # Validate the user input of option to visualize the answer in 3 Dimension
        while True:
            try:
                ask = input("Do you want to see the visualizer (Y/n): ").lower()
                if (ask != "y" and ask != "n"):
                    raise ValueError
            except ValueError:
                print("Please answer with yes (Y) or no (n)")
                continue
            else:
                return ask
                break
```

```
    def main():
        # Initialize answer string and file name
        answer = ""
        name = ""

        # ============================== INPUT SEGMENT ==============================
        print('''


 ____    ___ _____ ____    _____ ___ _   _ ____  _____ ____
|  _ \ / _ \_   _/ ___|  |  ___|_ _| \ | |  _ \| ____|  _ \
| | | | | | || | \___ \  | |_   | ||  \| | | | |  _| | |_) |
| |_| | |_| || |  ___) | |  _|  | || |\  | |_| | |___|  _ <
|____/ \___/ |_| |____/  |_|   |___|_| \_|____/|_____|_| \_\\


        ''')
        print("============================================================")
        print("------------ Welcome to the Closest Pair Finder ------------")

        print()
        neff = Main.askForDotNumber()
        print()
        dimension = Main.askForDimension()
        print()

        print("------------------------------------------------------------")
        print("============================================================")

        # ========================== CALCULATION SEGMENT ==========================
        # Generate the dots randomly
        dotList = ListDot(dimension, neff)

        # Bruteforce
        startTimeB = time.perf_counter()
        dot1, dot2, minDist1, euclidCount = dotList.bruteForce(0, dotList.neff - 1)
        endTimeB = time.perf_counter()

        # Divide and conquer
        startTimeD = time.perf_counter()
        dot3, dot4, minDist2 = dotList.divideAndConquer(0, dotList.neff - 1)
        endTimeD = time.perf_counter()

        # ============================== OUTPUT SEGMENT ==============================
        print("------------------------- Result --------------------------")

        answer += "Result using Bruteforce Algorithm:"
        answer += "\nFirst dot coordinate  : " + dot1.writeDot()
        answer += "\nSecond dot coordinate : " + dot2.writeDot()
        answer += f"\nMinimal distance      : {minDist1}"
        answer += f"\nEuclidean Distance function called {euclidCount} times"
        answer += f"\nAlgorithm execution time : {endTimeB-startTimeB} seconds"

        answer += "\n\nResult using Divide and Conquer Algorithm:"
        answer += "\nFirst dot coordinate  : " + dot3.writeDot()
        answer += "\nSecond dot coordinate : " + dot4.writeDot()
        answer += f"\nMinimal distance      : {minDist2}"
        answer += f"\nEuclidean Distance function called {dotList.euclid_count}
times"  answer += f"\nAlgorithm execution time : {endTimeD-startTimeD} seconds"
```

```
        print()
        print(answer)
        print()

        if (dimension == 3) :
            dotList.plotFor3D(dot1, dot2)

        print("-------------------------------------------------------------")
        print("=============================================================")
        # ================================ SAVING SEGMENT ================================
        print("------------------------- Saving ---------------------------")

        print()
        opt = Main.askForSaveToFile()
        if (opt == "y"):
            name = input("Please input the name of the file: ")
            IO.saveToFile(answer, name)

        print("\nThank you for using Closest Pair Finder")
        print("\nProcessed with : ")
        print("ASUS TUF GAMING F15")
        print("11th Gen Intel(R) Core(TM) i9-11900H @ 2.50GHz")


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

Gambar 2.4 Source Code File Main.py

BAB III

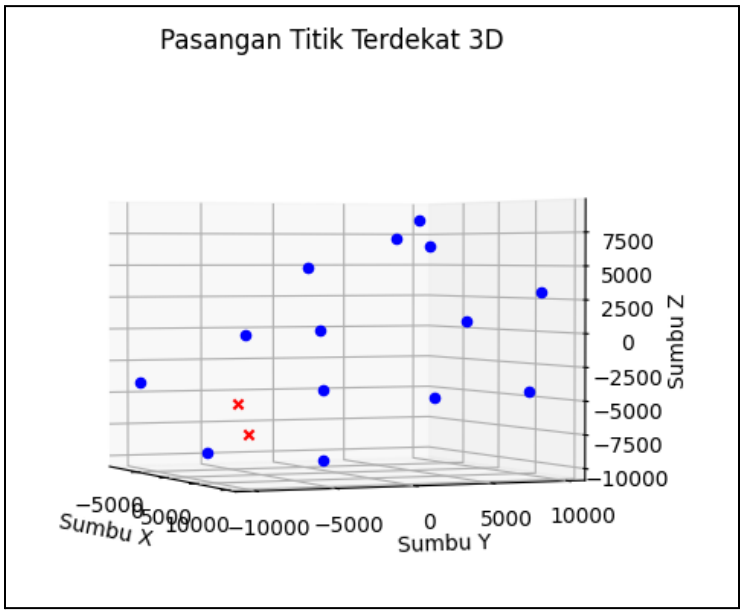
EKSPERIMEN DAN *TESTING*

3.1 *Testing* untuk Titik pada Ruang R^3

3.1.1 Banyak Titik = 16

```
Please input the number of dots : 16

Please input the number of dimension of the dot : 3

------------------------------------------------------------
============================================================
------------------------ Result ----------------------------

Result using Bruteforce Algorithm:
First dot coordinate  : [271.22, -6621.112, -5454.114]
Second dot coordinate : [3101.191, -7149.141, -7466.479]
Minimal distance      : 3512.429839713101
Euclidean Distance function called 120 times
Algorithm execution time : 0.0001067000002876739 seconds

Result using Divide and Conquer Algorithm:
First dot coordinate  : [271.22, -6621.112, -5454.114]
Second dot coordinate : [3101.191, -7149.141, -7466.479]
Minimal distance      : 3512.429839713101
Euclidean Distance function called 21 times
Algorithm execution time : 0.0001689000000624219 seconds

Do you want to see the visualizer (Y/n): Y
```

Gambar 3.1.1.1 Hasil pada 16 titik 3 dimensi

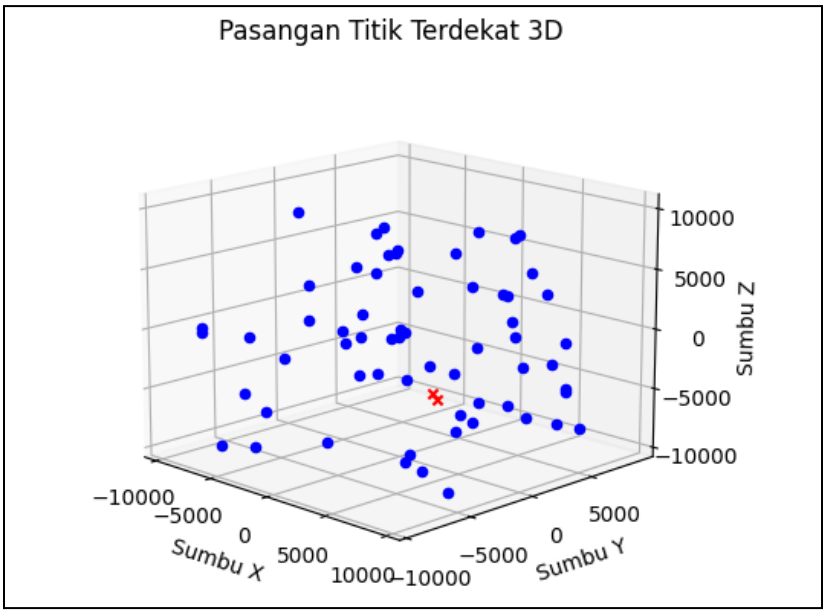


Gambar 3.1.1.2 Visualisasi hasil pada 16 titik 3 dimensi

3.1.2 Banyak Titik = 64

```
Please input the number of dots : 64

Please input the number of dimension of the dot : 3

------------------------------------------------------------
============================================================
------------------------ Result ----------------------------

Result using Bruteforce Algorithm:
First dot coordinate  : [4103.636, -1209.421, -3943.125]
Second dot coordinate : [4712.823, -1416.58, -4258.265]
Minimal distance      : 716.475311402982
Euclidean Distance function called 2016 times
Algorithm execution time : 0.0014249000005293055 seconds

Result using Divide and Conquer Algorithm:
First dot coordinate  : [4103.636, -1209.421, -3943.125]
Second dot coordinate : [4712.823, -1416.58, -4258.265]
Minimal distance      : 716.475311402982
Euclidean Distance function called 94 times
Algorithm execution time : 0.0006735999995726161 seconds

Do you want to see the visualizer (Y/n): Y
```

Gambar 3.1.2.1 Hasil pada 64 titik 3 dimensi

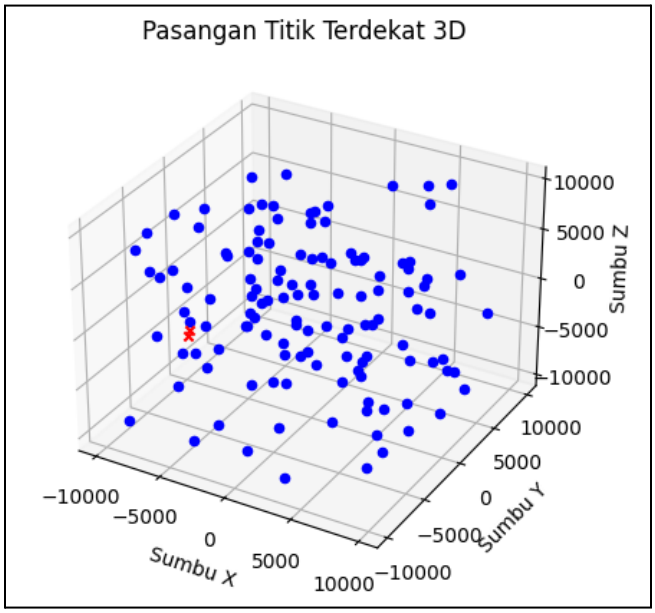


Gambar 3.1.2.2 Visualisasi hasil pada 64 titik 3 dimensi

3.1.3 Banyak Titik = 128

```
Please input the number of dots : 128

Please input the number of dimension of the dot : 3

------------------------------------------------------------
============================================================
----------------------- Result -----------------------------

Result using Bruteforce Algorithm:
First dot coordinate  : [-6435.446, -5072.087, -1165.881]
Second dot coordinate : [-6373.228, -5283.077, -1625.009]
Minimal distance      : 509.1035061831728
Euclidean Distance function called 8128 times
Algorithm execution time : 0.005840400000124646 seconds

Result using Divide and Conquer Algorithm:
First dot coordinate  : [-6435.446, -5072.087, -1165.881]
Second dot coordinate : [-6373.228, -5283.077, -1625.009]
Minimal distance      : 509.1035061831728
Euclidean Distance function called 159 times
Algorithm execution time : 0.0014193999995768536 seconds

Do you want to see the visualizer (Y/n): Y
```

Gambar 3.1.3.1 Hasil pada 128 titik 3 dimensi

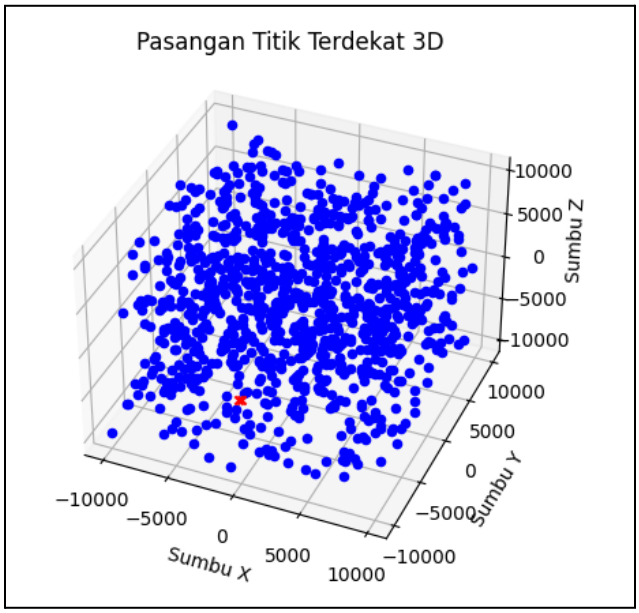


Gambar 3.1.3.2 Visualisasi hasil pada 128 titik 3 dimensi

3.1.4 Banyak Titik = 1000

```
Please input the number of dots : 1000

Please input the number of dimension of the dot : 3

------------------------------------------------------------
============================================================
------------------------- Result ---------------------------

Result using Bruteforce Algorithm:
First dot coordinate  : [-475.245, -9264.349, -2357.182]
Second dot coordinate : [-254.312, -9296.439, -2365.951]
Minimal distance      : 223.42348567238855
Euclidean Distance function called 499500 times
Algorithm execution time : 0.3658971000004385 seconds

Result using Divide and Conquer Algorithm:
First dot coordinate  : [-475.245, -9264.349, -2357.182]
Second dot coordinate : [-254.312, -9296.439, -2365.951]
Minimal distance      : 223.42348567238855
Euclidean Distance function called 1392 times
Algorithm execution time : 0.020973400000002584 seconds

Do you want to see the visualizer (Y/n): Y
```

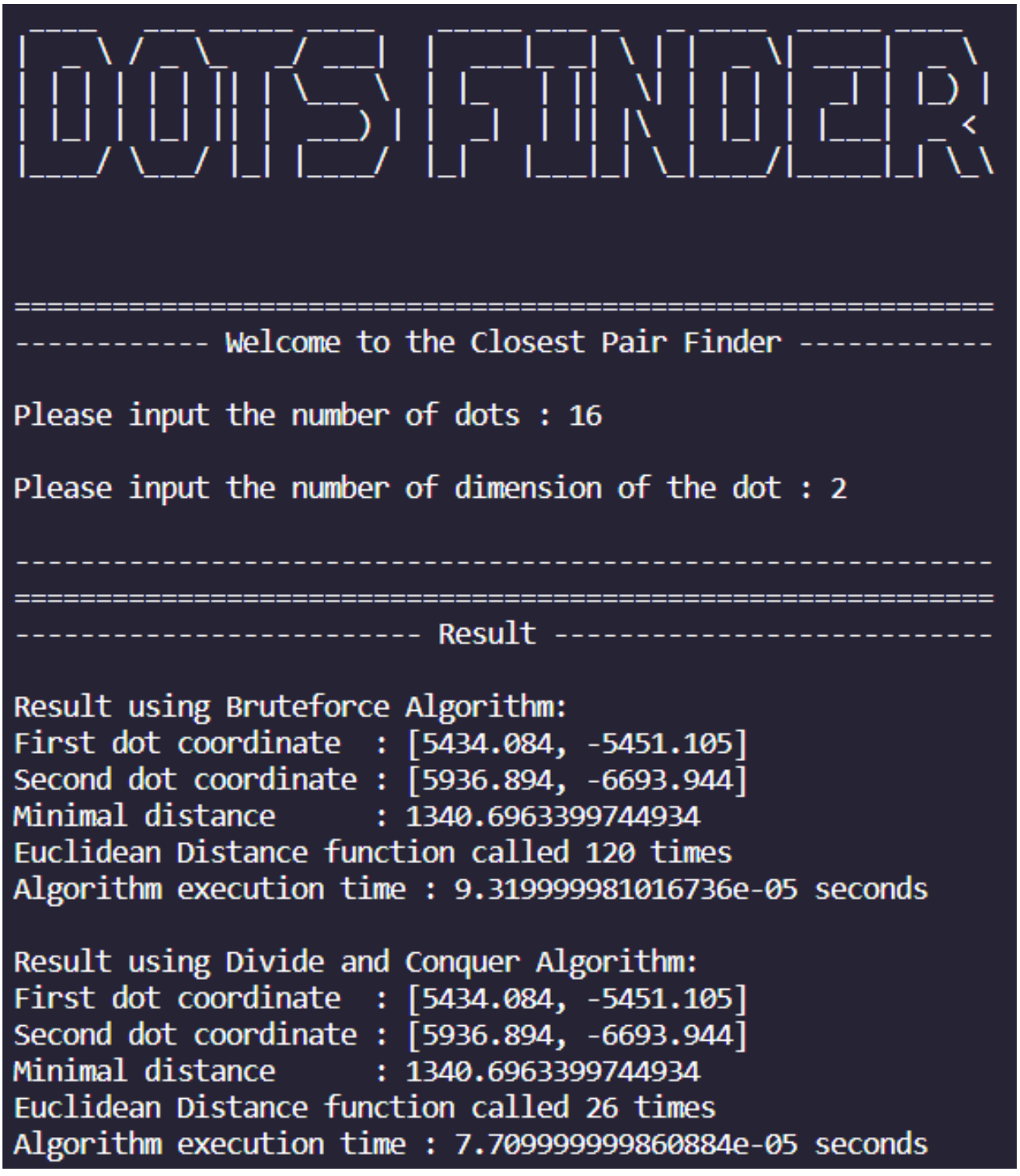
Gambar 3.1.4.1 Hasil pada 1000 titik 3 dimensi



Gambar 3.1.4.2 Visualisasi hasil pada 1000 titik 3 dimensi

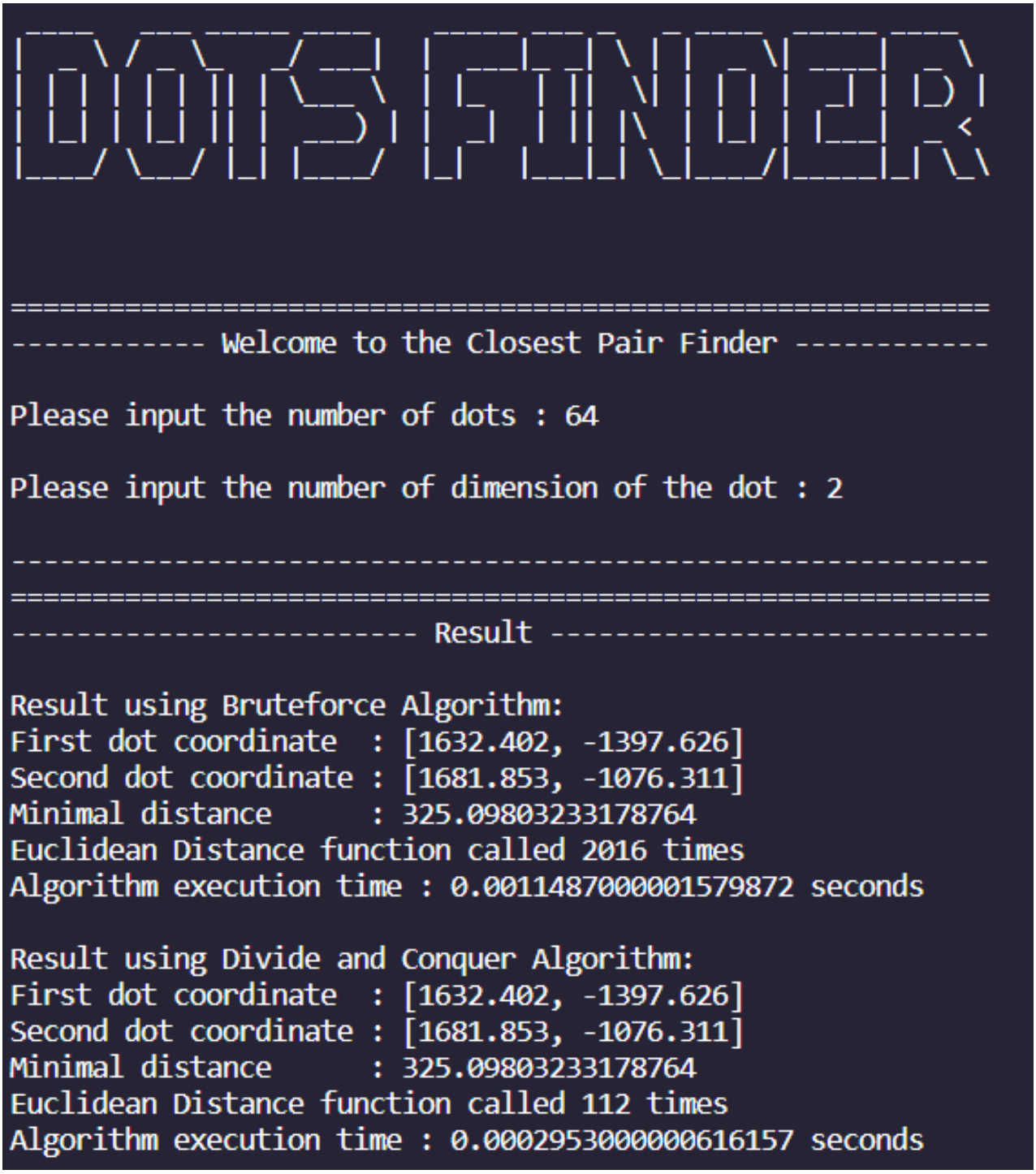
3.2 *Testing* untuk Titik pada Ruang R^n dengan $n \leq 2$

3.2.1 Banyak Titik = 16



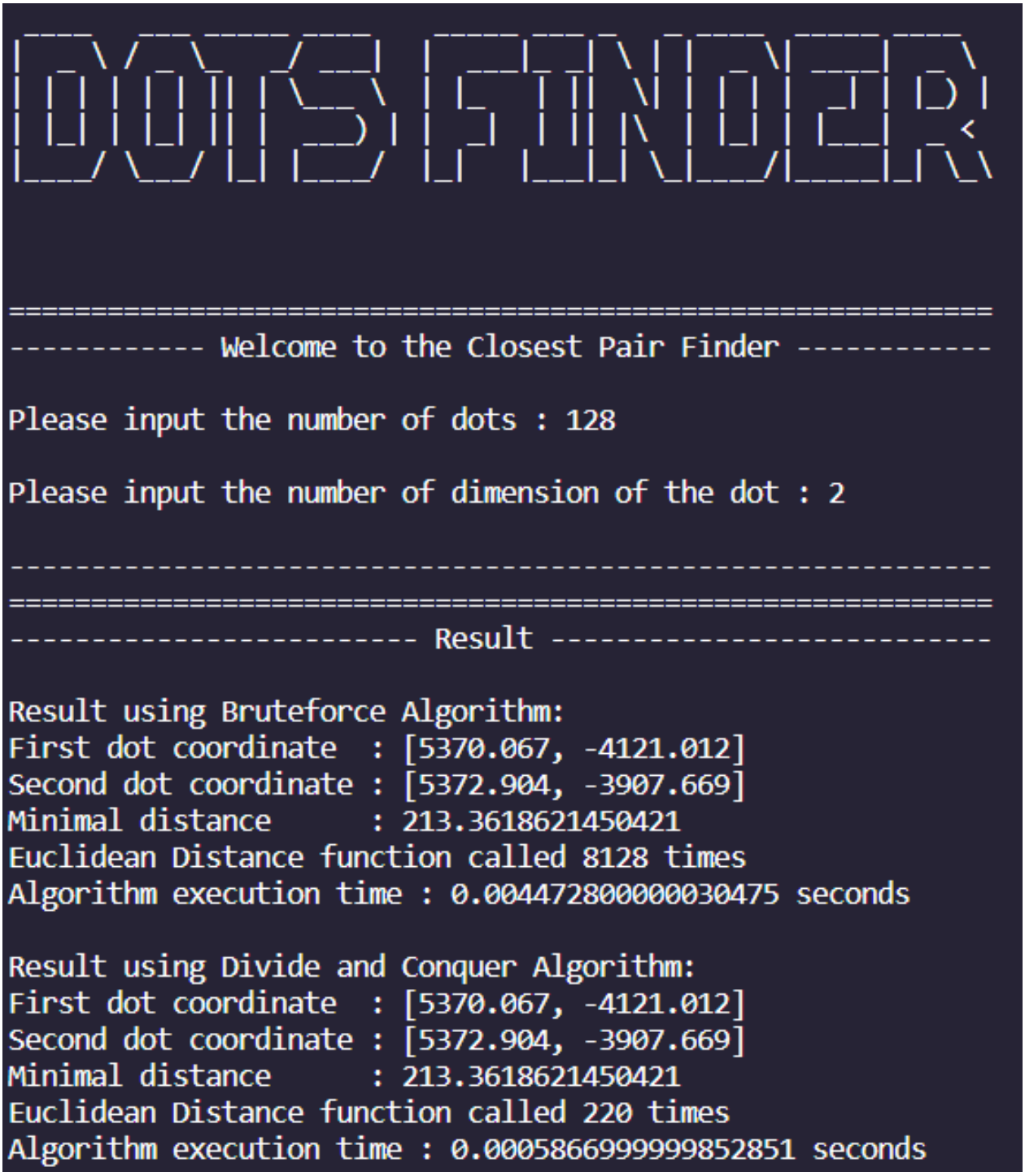
Gambar 3.2.1.1 Hasil pada 16 titik 2 dimensi

3.2.2 Banyak Titik = 64



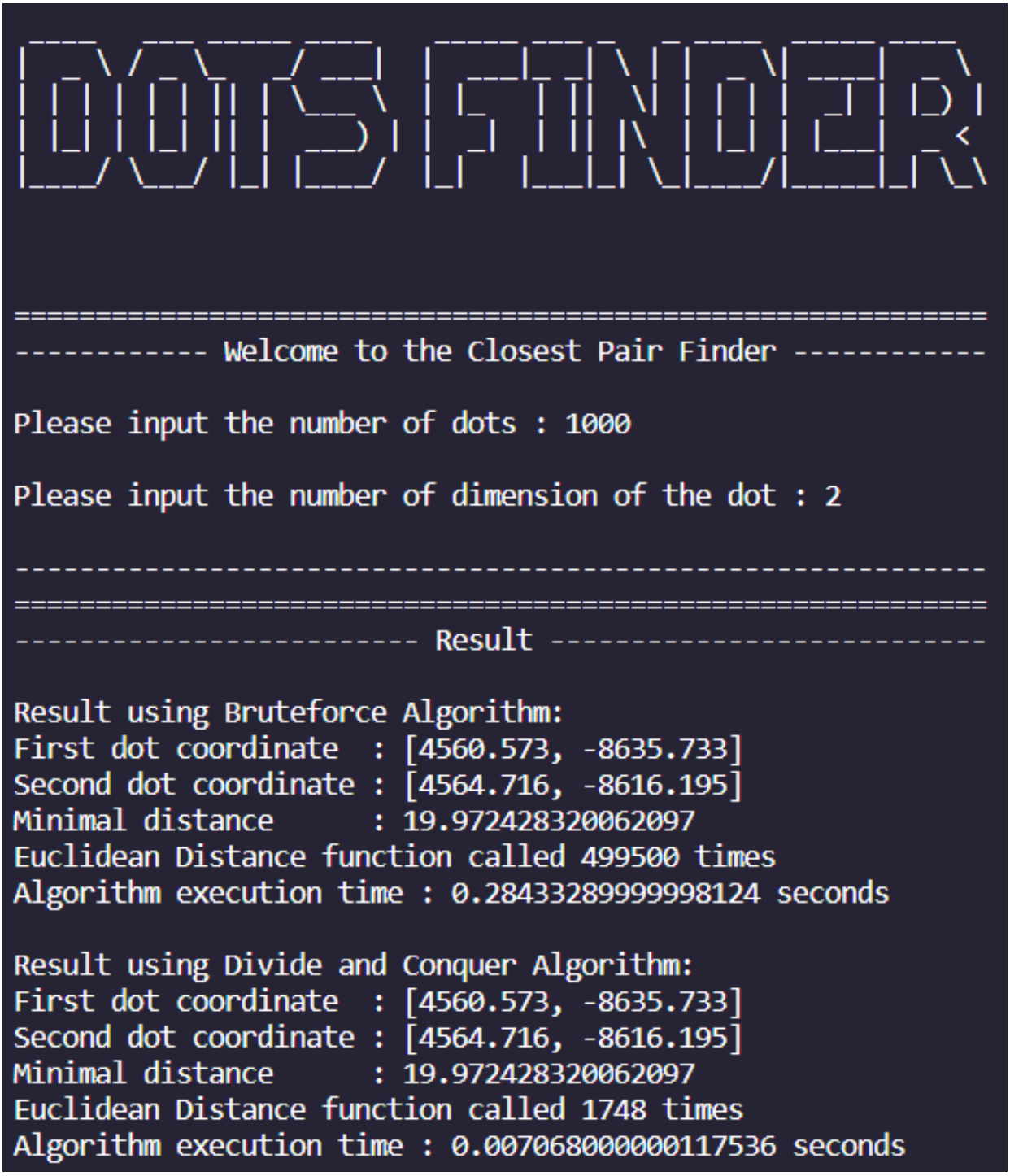
Gambar 3.2.2.1 Hasil pada 64 titik 2 dimensi

3.2.3 Banyak Titik = 128



Gambar 3.2.3.1 Hasil pada 128 titik 2 dimensi

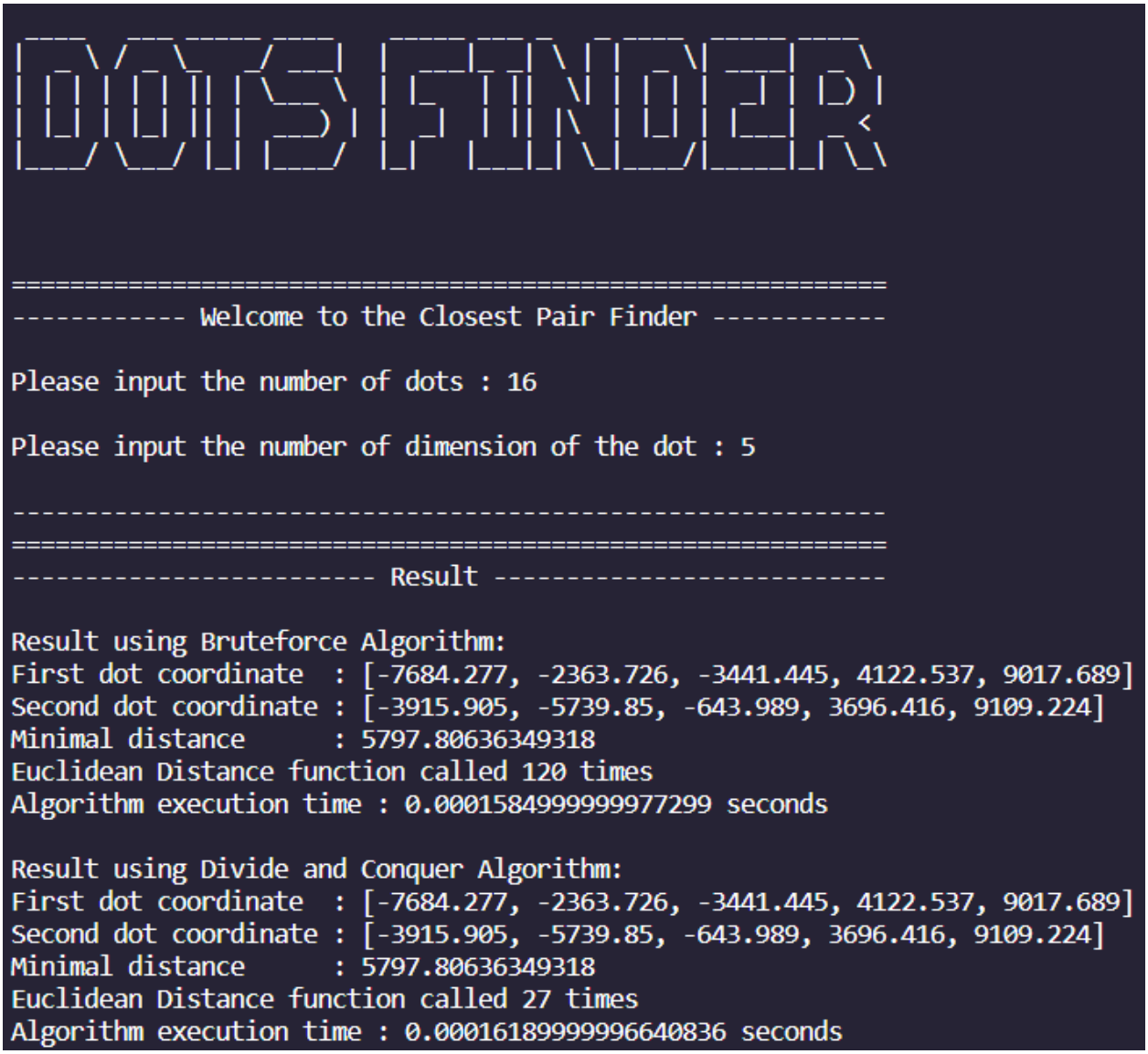
3.2.4 Banyak Titik = 1000



Gambar 3.2.4.1 Hasil pada 1000 titik 2 dimensi

3.3 *Testing* untuk Titik pada Ruang R^n dengan n > 3

3.3.1 Banyak Titik = 16



Gambar 3.3.1.1 Hasil pada 16 titik 5 dimensi

3.3.2 Banyak Titik = 64

```
 ____  ___ _____ ____    _____ ___ _   _ ____  _____ ____
|  _ \/ _ \_   _/ ___|  |  ___|_ _| \ | |  _ \| ____|  _ \
| | | | | || | \___ \  | |_   | ||  \| | | | |  _| | |_) |
| |_| | |_| || |  ___) | |  _| | || |\  | |_| | |___|  _ <
|____/ \___/ |_| |____/  |_|  |___|_| \_|____/|_____|_| \_\


============================================================
------------ Welcome to the Closest Pair Finder ------------

Please input the number of dots : 64

Please input the number of dimension of the dot : 5

------------------------------------------------------------
============================================================
------------------------- Result ---------------------------

Result using Bruteforce Algorithm:
First dot coordinate  : [-9230.469, -6477.93, -1186.979, -6528.367, -7345.603]
Second dot coordinate : [-8241.04, -4684.017, -1993.66, -5339.433, -7178.751]
Minimal distance      : 2507.8340215474786
Euclidean Distance function called 2016 times
Algorithm execution time : 0.002570600000126433 seconds

Result using Divide and Conquer Algorithm:
First dot coordinate  : [-9230.469, -6477.93, -1186.979, -6528.367, -7345.603]
Second dot coordinate : [-8241.04, -4684.017, -1993.66, -5339.433, -7178.751]
Minimal distance      : 2507.8340215474786
Euclidean Distance function called 137 times
Algorithm execution time : 0.000830700000278739 seconds
```

Gambar 3.3.2.1 Hasil pada 64 titik 5 dimensi

3.3.3 Banyak Titik = 128

```
Please input the number of dots : 128

Please input the number of dimension of the dot : 6

------------------------------------------------------------
============================================================
----------------------- Result -----------------------------

Result using Bruteforce Algorithm:
First dot coordinate  : [-5212.395, -1479.527, 6810.114, -3980.726, 5445.493
, -6764.193]
Second dot coordinate : [-4471.463, -1873.179, 4405.008, -3053.673, 6019.59,
 -9716.713]
Minimal distance      : 4049.0574184101165
Euclidean Distance function called 8128 times
Algorithm execution time : 0.008881499999915832 seconds

Result using Divide and Conquer Algorithm:
First dot coordinate  : [-5212.395, -1479.527, 6810.114, -3980.726, 5445.493
, -6764.193]
Second dot coordinate : [-4471.463, -1873.179, 4405.008, -3053.673, 6019.59,
 -9716.713]
Minimal distance      : 4049.0574184101165
Euclidean Distance function called 367 times
Algorithm execution time : 0.004208399999697576 seconds
```

Gambar 3.3.3.1 Hasil pada 128 titik 6 dimensi

3.3.4 Banyak Titik = 1000

```
Please input the number of dots : 1000

Please input the number of dimension of the dot : 7

------------------------------------------------------------
============================================================
------------------------- Result ---------------------------

Result using Bruteforce Algorithm:
First dot coordinate  : [954.814, -5456.627, 4204.991, 622.269, 8240.557, 65
05.906, 2926.417]
Second dot coordinate : [2211.702, -4482.688, 2236.755, -163.721, 8451.221,
7106.148, 2153.69]
Minimal distance      : 2832.2843515526474
Euclidean Distance function called 499500 times
Algorithm execution time : 0.6193967999997767 seconds

Result using Divide and Conquer Algorithm:
First dot coordinate  : [954.814, -5456.627, 4204.991, 622.269, 8240.557, 65
05.906, 2926.417]
Second dot coordinate : [2211.702, -4482.688, 2236.755, -163.721, 8451.221,
7106.148, 2153.69]
Minimal distance      : 2832.2843515526474
Euclidean Distance function called 3633 times
Algorithm execution time : 0.1198487000001478 seconds
```

Gambar 3.3.4.1 Hasil pada 1000 titik 7 dimensi

DAFTAR PUSTAKA

Munir, R. (2023). Algoritma *Divide and Conquer* (Bagian 2). IF2211 Strategi Algoritma - Semester II Tahun 2022/2023. Diakses pada 26 Februari 2023, pukul 10.43 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf

Steven. (2022). Closest pair of points in 3D. CS *Stack Exchange*. Diakses pada 22 Februari 2023, pukul 22.57 dari https://cs.stackexchange.com/questions/155286/closest-pair-of-points-in-3d

LAMPIRAN

Link Repository Program

GoDillonAudris512/Tucil2_13521062_13521084 (github.com)

Checklist Tugas Kecil

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