TUGAS KECIL II IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2022/2023

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

Disusun oleh:

Dhanika Novlisariyanti 13521132

Made Debby Almadea Putri 13521153

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

1. Algoritma Divide and Conquer ruang Rn **3**

2. Kode Program Dalam Bahasa Python **6**

2.1 Kelas Point 6

2.2 Kelas Points 7

2.3 Linalg 13

2.5 Input 15

2.6 Output 16

2.7 Main program 21

3. Contoh masukan dan luaran (skrinsut) **25**

a. Titik berjumlah 16 pada dimensi 3 dengan hasil titik randomize 25

Gambar 3.a.1 Hasil Eksekusi **26**

Gambar 3.a.2 Hasil Plot **26**

b. Titik berjumlah 64 pada dimensi 3 dengan hasil titik randomize 26

Gambar 3.b.1 Hasil Eksekusi **27**

Gambar 3.b.2 Hasil Plot **27**

c. Titik berjumlah 128 pada dimensi 3 dengan hasil titik randomize 27

Gambar 3.c.1 Hasil Eksekusi **28**

Gambar 3.c.2 Hasil Plot **28**

d. Titik berjumlah 1000 pada dimensi 3 dengan hasil titik randomize 28

Gambar 3.d.1 Hasil Eksekusi **29**

Gambar 3.d.2 Hasil Plot **29**

e. Titik berjumlah 16 pada dimensi 4 dengan hasil titik randomize 29

Gambar 3.e.1 Hasil Eksekusi **30**

f. Titik berjumlah 64 pada dimensi 5 dengan hasil titik randomize 30

Gambar 3.f.1 Hasil Eksekusi **30**

g. Titik berjumlah 16 pada dimensi 3 dengan input file 30

Gambar 3.g.1 Hasil Eksekusi **31**

Gambar 3.g.2 Hasil Eksekusi **31**

Gambar 3.g.3 Hasil Plot **31**

4. Pranala github **31**

5. Checklist **31**

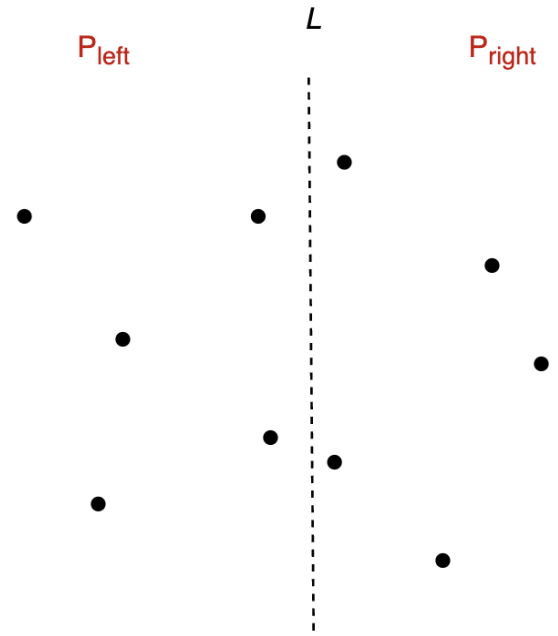
1. Algoritma Divide and Conquer ruang R^n

Misalkan terdapat himpunan titik *P* = $(p_1, p_2, \ldots, p_n)$ yang terdiri dari *n* buah titik pada ruang R^N $(x_1, x_2, \ldots, x_N)$. Untuk mendapatkan jarak terdekat dan semua pasang titik yang memiliki jarak terdekat dilakukan dengan algoritma sebagai berikut:

- *Preprocessing*: titik-titik di dalam *P* diurut secara menaik berdasarkan nilai x_1. Jika terdapat lebih dari satu titik dengan nilai x_1 yang sama, maka titik-titik tersebut diurutkan secara menaik berdasarkan nilai x_2 dan seterusnya(menggunakan algoritma *quicksort*)
- Algoritma *Finding Closest Pair*
  1. *SOLVE*: jika *n* = 1, maka tidak ada pasangan titik yang bisa dihitung jarak terdekatnya sehingga jarak terdekatnya adalah tidak terhingga. Jika *n* = 2, maka jarak terdekat adalah jarak kedua titik yang dihitung langsung dengan rumus *Euclidean*

$$\sqrt{(x_{1,a} - x_{1,b})^2 + (x_{2,a} - x_{2,b})^2 + \ldots + (x_{N,a} - x_{N,b})^2}$$

  2. *DIVIDE*: bagi himpunan titik *P* ke dalam dua bagian P_{left} dan P_{right}. Himpunan P_{left} merupakan himpunan bagian *P* yang berisi titik-titik $p_1, p_2, \ldots, p_{n/2}$. Sedangkan himpunan P_{right} merupakan himpunan bagian *P* yang berisi titik-titik $p_{n/2+1}, p_{n/2+2}, \ldots, p_n$. Misalkan terdapat sebuah garis maya *L* yang membagi titik-titik tersebut menjadi dua bagian



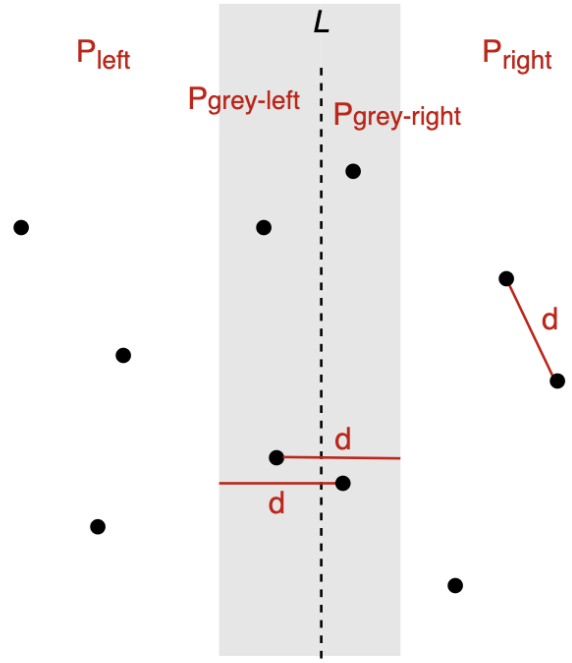
Gambar 1.1 Ilustrasi P_{left} dan P_{right}

3. *CONQUER*: Secara rekursif, diterapkan algoritma *Finding Closest Pair* pada masing-masing bagian sehingga menghasilkan jarak terdekat serta semua pasangan titik yang memiliki jarak terdekat pada bagian tersebut
4. *COMBINE*: Terdapat 3 kemungkinan letak pasangan titik terdekat
   a) Pasangan titik terdekat terdapat pada bagian P_{left}
   b) Pasangan titik terdekat terdapat pada bagian P_{right}
   c) Pasangan titik terdekat merupakan titik yang berada pada bagian P_{left} dengan titik yang berada pada bagian P_{right}

   Pada tahap ini, kita bandingkan pasangan titik terdekat pada bagian P_{left} dengan pasangan titik terdekat pada bagian P_{right} dan mencari pasangan titik yang memiliki jarak terdekat antara kedua pasangan titik tersebut. Jika kedua pasangan titik memiliki jarak yang sama, maka kedua pasangan titik tersebut merupakan solusi sementara.

   Pasangan titik tersebut masih solusi sementara karena terdapat kemungkinan c. Oleh karena itu, diperlukan tahap tambahan untuk mencari jarak terdekat dua buah titik dan semua pasang titik yang memiliki jarak terdekat sebagai solusi dari persoalan.

Misalkan d merupakan jarak terdekat dari solusi sementara, maka untuk mengatasi kasus c, dilakukan langkah-langkah sebagai berikut:



Gambar 1.1 Ilustrasi $P_{grey-left}$ *dan* $P_{grey-right}$

1. Mencari titik-titik di P_{left} yang memiliki x_1 minimal $x_{1,n/2+1} - d \,.\, x_{1,n/2+1}$ ini merupakan elemen P_{right} yang memiliki jarak terdekat dengan garis maya L. Sebut titik-titik yang ditemukan sebagai himpunan $P_{gray-left}$
2. Mencari titik-titik di P_{right} yang memiliki x_1 maksimal $x_{1,n/2} + d.\, x_{1,n/2}$ ini merupakan elemen P_{left} yang memiliki jarak terdekat dengan garis maya L. Sebut titik-titik yang ditemukan sebagai himpunan $P_{gray-right}$
3. Untuk setiap titik pada himpunan $P_{gray-left}$ dihitung jaraknya dengan setiap titik pada himpunan $P_{gray-right}$. Akan tetapi, jika salah satu nilai $x_1, x_2, \ldots, x_N$ dari kedua titik memiliki selisih yang lebih besar dari d, maka perhitungan jarak tidak perlu dilakukan karena sudah pasti memiliki jarak yang lebih besar dari d.

Pada setiap pasangan titik, jika jarak antara kedua titik lebih kecil dari *d*, maka kita perbaharui *d* menjadi jarak antara kedua titik tersebut (solusi sementara yang baru). Jika jarak antara kedua titik sama dengan *d*, maka pasangan titik tersebut merupakan alternatif solusi pasangan titik terdekat

2. Kode Program Dalam Bahasa Python

2.1 Kelas Point

```python
import random


class Point:
    def __init__(self, dimension, coordinate=[]):
        self.dimension = dimension
        if (len(coordinate) == 0):
            coordinate = [0 for i in range(dimension)]
        self.coordinate = coordinate

    def get(self, axis):
        return self.coordinate[axis]

    def set(self, axis, value):
        self.coordinate[axis] = value

    def generate_random(self, constraint):
        """
        menghasilkan koordinat secara random (double)
        dengan batas -constraint hingga constraint (inklusif)
        """
        for i in range(self.dimension):
            self.set(i, random.uniform(-constraint, constraint))

    def is_diff_within_distance(self, other, distance):
        """
        return true jika selisih self dan other
        pada setiap sumbu kurang dari sama dengan distance
        """
        for i in range(self.dimension):
            if abs(self.get(i) - other.get(i)) > distance:
                return False

        return True

    def greater_than_eq(self, other, axis=0):
        """
        return true jika self lebih besar sama dengan other
        dengan prioritas utama sumbu axis
        """
        if self.get(axis) > other.get(axis):
            return True
        elif self.get(axis) < other.get(axis):
```

```
            return False

        for i in range(self.dimension):
            if axis == i:
                continue
            if self.get(i) > other.get(i):
                return True
            elif self.get(i) < other.get(i):
                return False

        return True

    def less_than_eq(self, other, axis=0):
        """
        return true jika self lebih kecil sama dengan other
        dengan prioritas utama sumbu axis
        """
        if self.get(axis) < other.get(axis):
            return True
        elif self.get(axis) > other.get(axis):
            return False

        for i in range(self.dimension):
            if axis == i:
                continue
            if self.get(i) < other.get(i):
                return True
            elif self.get(i) > other.get(i):
                return False

        return True
```

2.2 Kelas Points

```
import points.point as point
import lib.linalg as la
import math
import time


class Points:
    def __init__(self, dimension, points=[]):
        self.__dimension = dimension
        self.__points = points
        self.__point_count = 0

    # getter
    def get_points(self):
```

```
        """
            return points
        """
        return self.__points

    def get_point(self, id):
        return self.__points[id]

    def get_points_within_id(self, start_id, end_id):
        """
            return points dari start_id s.d. end_id
        """
        if start_id <= end_id:
            return self.__points[start_id:end_id + 1]
        else:
            return []

    def get_dimension(self):
        return self.__dimension

    def get_point_count(self):
        return self.__point_count

    # setter
    def set_points(self, points):
        self.__points = points
        self.__point_count = len(points)

    def set_empty(self):
        self.__points = []
        self.__point_count = 0

    # other function

    def generate_random(self, point_count, constraint):
        """
            menghasilkan points secara random
        """
        self.set_empty()
        _points = []
        for i in range(point_count):
            _point = point.Point(self.get_dimension())
            _point.generate_random(constraint)
            _points.append(_point)
        self.set_points(_points)

    def add(self, point):
        """
            tambahkan point ke dalam points
        """
        self.__points.append(point)
        self.__point_count += 1

    def __partition(self, lowIdx, highIdx, axis):
```

```
        """
            partisi untuk quicksort
        """
        # get 1-3
        pivot = self.get_point(highIdx)  # Pivot x terakhir
        i = lowIdx - 1
        for k in range(lowIdx, highIdx):
            if self.get_point(k).less_than_eq(pivot, axis):
                i += 1
                self.__points[i], self.__points[k] = self.__points[k],
self.__points[i]

        self.__points[i+1], self.__points[highIdx] = self.__points[highIdx],
self.__points[i+1]
        return i+1

    def sort(self, lowId, highIdx, axis=0):
        """
            mengurutkan points berdasarkan axis
        """
        i = lowId
        j = highIdx
        if i < j:
            partitionId = self.__partition(lowId, highIdx, axis)
            self.sort(i, partitionId-1, axis)
            self.sort(partitionId+1, j, axis)

    def __search_fo(self, low, high, value, axis):
        """
            prereq: points terurut membesar berdasarkan axis

            return indeks dari kejadian pertama ditemukan nilai dari axis
            sebesar value dengan menggunakan binary search
            jika tidak ada nilai dari axis yang bernilai value, maka
            return indeks kejadian pertama ditemukannya nilai dari axis
            lebih besar dari value
        """
        if (high >= low and low < self.__point_count):
            mid = low + (high - low) // 2
            if self.get_point(mid).get(axis) == value:
                if mid == 0 or self.get_point(mid - 1).get(axis) < value:
                    return mid
                else:
                    return self.__search_fo(low, high - 1, value, axis)
            else:
                if value < self.get_point(mid).get(axis):
                    return self.__search_fo(low, mid - 1, value, axis)
                else:
                    return self.__search_fo(mid + 1, high, value, axis)

        return low

    def __search_lo(self, low, high, value, axis):
        """
```

```
            prereq: points terurut membesar berdasarkan axis

            return indeks dari kejadian terakhir ditemukan nilai dari axis
            sebesar value dengan menggunakan binary search
            jika tidak ada nilai dari axis yang bernilai value, maka
            return indeks kejadian terakhir ditemukannya nilai dari axis
            lebih kecil dari value
        """
        if (high >= low and high < self.__point_count):
            mid = low + (high - low) // 2
            if self.get_point(mid).get(axis) == value:
                if mid == self.get_point_count() - 1 or self.get_point(mid +
1).get(axis) > value:
                    return mid
                else:
                    return self.__search_lo(low, high + 1, value, axis)
            else:
                if value < self.get_point(mid).get(axis):
                    return self.__search_lo(low, mid - 1, value, axis)
                else:
                    return self.__search_lo(mid + 1, high, value, axis)

        return high

    def __search(self, value, kind="first", axis=0):
        """
            prereq: points terurut membesar berdasarkan axis
        """
        if kind == "first":
            return self.__search_fo(0, self.get_point_count() - 1, value, axis)
        elif kind == "last":
            return self.__search_lo(0, self.get_point_count() - 1, value, axis)

    def divide(self):
        """
            return tuple
            1. array of integer dengan dua elemen merepresentasikan indeks awal
               dan indeks akhir dari setengah bagian awal points
            2. array of integer dengan dua elemen merepresentasikan indeks awal
               dan indeks akhir dari setengah bagian akhir points
        """
        left_sid = 0
        left_eid = self.__point_count // 2 - 1
        right_sid = self.__point_count // 2
        right_eid = self.__point_count - 1

        return [left_sid, left_eid], [right_sid, right_eid]

    def __find_closest_pair_grey(self, distance, left_id, right_id, axis=0):
        """
            return pasangan point di dalam grey area
        """
        _min = distance
        result = []
```

```
        left_closest = self.get_point(left_id[1])
        right_closest = self.get_point(right_id[0])

        # grey area farthest left id
        gfl_id = self.__search(
            right_closest.get(axis) - distance, kind="first")

        grey_l = Points(self.__dimension)
        grey_l.set_points(self.get_points_within_id(gfl_id, left_id[1]))

        if grey_l.get_point_count() > 0:
            # grey area farthest right id
            gfr_id = self.__search(left_closest.get(
                axis) + distance, kind="last")
            grey_r = Points(self.__dimension)
            grey_r.set_points(self.get_points_within_id(right_id[0],  gfr_id))

            for i in range(grey_l.__point_count):
                p1 = grey_l.get_point(grey_l.get_point_count() - i - 1)
                for i in range(grey_r.__point_count):
                    p2 = grey_r.__points[i]

                    if not p2.is_diff_within_distance(p1, _min):
                        continue
                    _norm = la.norm(p1, p2)
                    if _norm < _min:
                        _min = _norm
                        result = [[p1, p2]]
                    elif _norm == _min:
                        result += [[p1, p2]]

        if len(result) > 0:
            return _min, result
        else:
            return math.inf, []

    def __find_closest_pair_dnc(self):
        """
            mencari closest pair of points dengan algoritma divide and conquer
        """
        if self.get_point_count() == 1:
            return math.inf, []
        elif self.get_point_count() == 2:
            norm = la.norm(self.get_point(0), self.get_point(1))
            return norm, [[self.get_point(0), self.get_point(1)]]
        else:
            _min = 0
            result = []

            left_id, right_id = self.divide()
            left = Points(self.__dimension)
            left.set_points(self.get_points_within_id(left_id[0], left_id[1]))
```

```
            right = Points(self.__dimension)
            right.set_points(self.get_points_within_id(
                right_id[0], right_id[1]))

            left_min, left_result = left.__find_closest_pair_dnc()
            right_min, right_result = right.__find_closest_pair_dnc()

            if (left_min < right_min):
                _min = left_min
                result = left_result
            elif (left_min > right_min):
                _min = right_min
                result = right_result
            else:
                _min = left_min
                result = left_result + right_result

            _min_grey, grey_result = self.__find_closest_pair_grey(
                _min, left_id, right_id)

            if _min_grey < _min:
                _min = _min_grey
                result = grey_result
            elif _min_grey == _min:
                result += grey_result

            return _min, result

    def __find_closest_pair_bf(self):
        """
            mencari closest pair of points dengan algoritma brute force
        """
        _min = math.inf
        result = []
        for i in range(self.__point_count):
            for j in range(i + 1, self.__point_count):
                _norm = la.norm(self.get_point(i), self.get_point(j))
                if (_norm < _min):
                    _min = _norm
                    result = [[self.get_point(i), self.get_point(j)]]
                elif (_norm == _min):
                    result += [[self.get_point(i), self.get_point(j)]]

        return _min, result

    def find_closest_pair(self, kind="dnc"):
        """
            mencari closest pair of points.

            kind: algoritma yang digunakan, dapat berupa "dnc" atau "bf"
        """
        if kind == "bf":
            start = time.perf_counter()
            _min, result = self.__find_closest_pair_bf()
```

```
            end = time.perf_counter()
            return _min, result, end - start
        elif kind == "dnc":
            self.sort(0, self.get_point_count() - 1)
            start = time.perf_counter()
            _min, result = self.__find_closest_pair_dnc()
            end = time.perf_counter()
            return _min, result, end - start

    def view(self):
        """
            mencetak semua point dalam points
        """
        for i in range(self.__point_count):
            print(self.get_point(i).coordinate)
```

2.3 Linalg

```
import math

# menghitung pemanggilan fungsi norm
func_called = 0


def norm(point1, point2):
    """
    mengembalikan jarak euclidean
    dari point1 dan point2
    """
    global func_called
    func_called += 1

    norm = 0
    for i in range(point1.dimension):
        norm += math.pow(point1.get(i) - point2.get(i), 2)

    return math.sqrt(norm)
```

2.4 Visualizer

```
import matplotlib.pyplot as plt
```

```
def isPointResult(point, result):
    """
    mengembalikan true jika point ada di dalam result
    """
    for i in range(len(result)):
        if point.is_equal(result[i][0]) or point.is_equal(result[i][1]):
            return True

    return False


def visualize(points, pairedPoints, fileName):
    """
    Plotting points
    """
    colorArr = ["red", "blue", "green", "cyan", "yellow",
                "orange", "green", "purple", "pink"]
    colorId = 0
    ax = plt.figure(figsize=(15, 10)).add_subplot(111, projection='3d')

    for i in range(points.get_point_count()):
        _point = points.get_point(i)
        if isPointResult(_point, pairedPoints):
            continue
        else:
            ax.scatter(_point.get(0), _point.get(
                1), _point.get(2), color="gray")

    for i in range(len(pairedPoints)):
        ax.scatter(pairedPoints[i][0].get(0), pairedPoints[i][0].get(
            1), pairedPoints[i][0].get(2), color=colorArr[colorId])
        ax.scatter(pairedPoints[i][1].get(0), pairedPoints[i][1].get(
            1), pairedPoints[i][1].get(2), color=colorArr[colorId])
        if colorId == len(colorArr) - 1:
            colorId = 0
        else:
            colorId += 1
    """
    Modify Graph
    """
    plt.title("Find Closest Distance 3D")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.clabel("z")
    ax.xaxis.label.set_color('red')
    ax.yaxis.label.set_color('blue')
    ax.w_xaxis.line.set_color("red")
    ax.w_yaxis.line.set_color("blue")
    ax.w_zaxis.line.set_color("green")

    # Show
    plt.savefig('output/' + fileName)
```

```
    plt.show()
```

2.5 Input

```
import points.points as ps
import points.point as p


def inputFile(inputFileName):
    """
    membaca input dari file
    """
    f = open("input/" + inputFileName + ".txt", "r")
    readText = []
    readText = f.read()
    f.close()
    splitText = readText.splitlines()

    if len(splitText) < 2:
        raise Exception("Incorrect File configuration")

    dimension = int(splitText[0])
    if dimension > 100:
        raise Exception(
            "Incorrect File configuration: Dimension greater than 100. Too
big")

    numberOfPoints = int(splitText[1])
    if numberOfPoints < 2:
        raise Exception(
            "Incorrect File configuration: Number of points are less than 2")

    count = len(splitText) - 2

    if count != numberOfPoints:
        raise Exception("Incorrect File configuration: Unmatched number of
points. Expected: " +
                        str(numberOfPoints) + ". Obtained: " + str(count))

    for i in range(2, len(splitText)):
        for j in range(dimension):
            split = splitText[i].split()
            if (len(split) != dimension):
                raise Exception(
                    "Incorrect File configuration: Incorrect dimension for
point in line " + str(i + 1))

    return splitText, dimension, numberOfPoints
```

```
def processPoints(splitText, dimension, numberOfPoints):
    """
    mengubah input array of integer menjadi points
    """
    hasil = [0 for i in range(numberOfPoints)]
    for i in range(2, numberOfPoints+2):
        hasil[i-2] = splitText[i].split()

    for i in range(numberOfPoints):
        for j in range(dimension):
            hasil[i][j] = float(hasil[i][j])
            if hasil[i][j] > 1e9:
                raise Exception(
                    "Incorrect File configuration: There is a point with more
than 1e9 value")

    ps1 = ps.Points(dimension)
    for i in range(numberOfPoints-2):
        ps1.add(p.Point(dimension, hasil[i]))
    return ps1
```

2.6 Output

```
import platform
import lib.linalg as la
import sys
import colorama
from colorama import Fore
import psutil

# To reset color
colorama.init(autoreset=True)


def printWelcome():
    print(Fore.YELLOW + """
                __          __        .__
         /  \    /  \ ____ |  |   ____ ____   _____   ____
         \   \/\/   // __ \|  | _/ ___\/  _ \ /     \_/ __ /
          \        /\  ___/|  |_\  \__(  <_> )  Y Y  \  ___/
           \__/\  /  \___  >____/\___  >____/|__|_|  /\___  >
                \/       \/          \/            \/     \/
              """)
    print()
```

```
def printPlatform():
   print("""
           Here is your computer specification:
           'platform': {},
           'platform-release': {},
           'platform-version': {},
           'architecture': {},
           'processor': {},
           'ram': {} GB
           """.format(platform.system(), platform.release(),
platform.version(), platform.machine(), platform.processor(),
(psutil.virtual_memory()[0]/1000000000)))


def printPlatformToFile():
   print("Here is your computer specification:")
   print("'platform': {}".format(platform.system()))
   print("'platform-release': {}".format(platform.release()))
   print("'architecture': {}".format(platform.machine()))
   print("'processor': {}".format(platform.processor()))
   print("'ram': {} GB".format(psutil.virtual_memory()[0]/1000000000))
   print("")


def printMenu():
   print("")
   print(Fore.LIGHTGREEN_EX +

"-----------------------------------------Menu-------------------------------
------------------")
   print(Fore.LIGHTGREEN_EX +
         "|1. Randomize Points
|")
   print(Fore.LIGHTGREEN_EX +
         "|2. Input File
|")
   print(Fore.LIGHTGREEN_EX +
         "|3. Exit
|")
   print(Fore.LIGHTGREEN_EX +

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


def printDash():
   print(Fore.RED +
"--------------------------------------------------------------------------------
------------------")


def result(_points):
   """
```

```
    mencari closest pair
    """
    la.func_called = 0
    _minDNC, resultDNC, finalTimeDNC = _points.find_closest_pair()
    calledDNC = la.func_called

    la.func_called = 0
    _minBF, resultBF, finalTimeBF = _points.find_closest_pair(kind="bf")
    calledBF = la.func_called

    return _points, _minDNC, resultDNC, calledDNC, finalTimeDNC, _minBF,
resultBF, calledBF, finalTimeBF


def printToTerminal(_minDNC, resultDNC, calledDNC, finalTimeDNC, _minBF,
resultBF, calledBF, finalTimeBF):
    print("")
    print(Fore.GREEN +
"-------------------------------------------------------------------------
------------------")
    print(Fore.GREEN + "|  Algorithm  |     Minimum Distance       |      Time
|      Function Norm Called      |")
    print(Fore.GREEN +
"-------------------------------------------------------------------------
------------------")
    print(Fore.GREEN + "|    DNC      |          {:.2f}      |     {:.4f} s
|             {}              |".format(
        _minDNC, finalTimeDNC, calledDNC))
    print(Fore.GREEN +
"-------------------------------------------------------------------------
------------------")
    print(Fore.GREEN + "|    BF       |          {:.2f}      |     {:.4f} s
|             {}              |".format(
        _minBF, finalTimeBF, calledBF))
    print(Fore.GREEN +
"-------------------------------------------------------------------------
------------------")
    print("")

    print("")
    print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
    print(Fore.CYAN + "|                                          Closest Pair
DNC                                  |")
    print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
    for i in range(len(resultDNC)):
        print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
        print(Fore.CYAN + "|                                            Closest
Pair {}                                |".format(i+1))
```

```
        print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
        for j in range(len(resultDNC[i])):
            print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
            print(Fore.CYAN + "|{}|".format(resultDNC[i][j].coordinate))
            print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
    print("")

    print("")
    print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
    print(Fore.CYAN + "|                                          Closest Pair BF
|")
    print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
    for i in range(len(resultBF)):
        print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
        print(Fore.CYAN + "|                                              Closest
Pair {}                                |".format(i+1))
        print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
        for j in range(len(resultBF[i])):
            print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
            print(Fore.CYAN + "|{}|".format(resultBF[i][j].coordinate))
            print(Fore.CYAN +
"-------------------------------------------------------------------------
------------------")
    print("")


def printToFile(_minDNC, resultDNC, calledDNC, finalTimeDNC, _minBF, resultBF,
calledBF, finalTimeBF):
    print("")

print("-------------------------------------------------------------------------
------------------------")
    print("|  Algorithm  |     Minimum Distance       |      Time       |
Function Norm Called      |")

print("-------------------------------------------------------------------------
------------------------")
    print("|    DNC      |         {:.2f}      |     {:.4f} s       |
```

```
{}              |".format(
        _minDNC, finalTimeDNC, calledDNC))

print("-------------------------------------------------------------------------
------------------------")
    print("|    BF       |          {:.2f}      |     {:.4f} s       |
{}              |".format(
        _minBF, finalTimeBF, calledBF))

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

    print("")

print("-------------------------------------------------------------------------
------------------------")
    print("|                                          Closest Pair DNC
|")

print("-------------------------------------------------------------------------
------------------------")
    for i in range(len(resultDNC)):

print("-------------------------------------------------------------------------
------------------------")
        print("|                                             Closest Pair {}
|".format(i+1))

print("-------------------------------------------------------------------------
------------------------")
        for j in range(len(resultDNC[i])):

print("-------------------------------------------------------------------------
------------------------")
            print("|{}|".format(resultDNC[i][j].coordinate))

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

    print("")

print("-------------------------------------------------------------------------
------------------------")
    print("|                                          Closest Pair BF
|")

print("-------------------------------------------------------------------------
------------------------")
    for i in range(len(resultBF)):

print("-------------------------------------------------------------------------
------------------------")
```

```
        print("|                                                    Closest Pair {}
|".format(i+1))

print("-------------------------------------------------------------------------
------------------------")
        for j in range(len(resultBF[i])):

print("-------------------------------------------------------------------------
------------------------")
            print("|{}|".format(resultBF[i][j].coordinate))

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


def outputToFile(outputFileName, dimension, pointCount, _minDNC, resultDNC,
calledDNC, finalTimeDNC, _minBF, resultBF, calledBF, finalTimeBF):
    original_stdout = sys.stdout
    with open("output/" + outputFileName+".txt", 'w') as f:
        sys.stdout = f
        printPlatformToFile()
        print("Dimension: {}".format(dimension))
        print("n: {}".format(pointCount))
        printToFile(_minDNC, resultDNC, calledDNC,
                    finalTimeDNC, _minBF, resultBF, calledBF, finalTimeBF)

        sys.stdout = original_stdout

        f.close()


def printGoodbye():
    print(Fore.YELLOW + """
             ________                    .______.
            /  _____/  ____   ____   __| _/\_ |__ ___.__. ____
            /   \  ___ /  _ \ /  _ \ / __ |  | __ <   |  |/ __ \
            \    \_\  (  <_> |  <_> ) /_/ |  | \_\ \___  \  ___/
            \______  /\____/ \____/\____ |  |___  / ____|\___  >
                   \/                   \/      \/\/         \/
              """)
```

2.7 Main program

```
import visualizer.visualizer as vs
import utils.output as op
import utils.input as ip
import points.points as ps
```

```
if __name__ == "__main__":
    start = True
    while (start):
        op.printWelcome()
        op.printPlatform()
        op.printMenu()
        inputMenu = input("Choice: ")
        op.printDash()

        if (inputMenu == "1"):

            startPoint = True

            while (startPoint):

                print("Input dimension or type 'e' to go back to menu")
                r_n = input("Dimension size: ")

                if (r_n.isdigit() and int(r_n) <= 100):
                    countPoint = input("Input n points: ")
                    if (countPoint.isdigit() and int(countPoint) > 1 and
int(countPoint) <= 10000):
                        CONSTRAINT = 1e9
                        _points = ps.Points(int(r_n))
                        _points.generate_random(int(countPoint), CONSTRAINT)
                        _points, _minDNC, resultDNC, calledDNC, finalTimeDNC,
_minBF, resultBF, calledBF, finalTimeBF = op.result(
                            _points)
                        op.printToTerminal(
                            _minDNC, resultDNC, calledDNC, finalTimeDNC,
_minBF, resultBF, calledBF, finalTimeBF)

                        toFile = input("Print to File? y/n ")

                        while toFile != "y" and toFile != "Y" and toFile != "n"
and toFile != "N":
                            print("Please input between y or n")
                            print(
                                "-------------------------------------")
                            toFile = input("Print to File? y/n ")

                        if (toFile == "y" or toFile == "Y"):
                            fileName = input("File Name: ")
                            op.outputToFile(fileName, r_n, countPoint, _minDNC,
resultDNC,
                                            calledDNC, finalTimeDNC, _minBF,
resultBF, calledBF, finalTimeBF)

                        elif (toFile == "n" or toFile == "N"):
                            findFile = False

                        if(int(r_n) == 3):
```

```
                            print("Do you want to visualize the data? y/n")
                            inputVisualize = input("Choice: ")
                            while inputVisualize != "y" and inputVisualize !=
"Y" and inputVisualize != "n" and inputVisualize != "N":
                                print("Please input between y or n")
                                print(
                                    "-------------------------------------")
                                inputVisualize = input("Choice: ")

                            if (inputVisualize == "y" or inputVisualize ==
"Y"):
                                fileName = input("Figure Name: ")
                                print("Visualizing...")
                                vs.visualize(_points, resultDNC, fileName)
                            elif (inputVisualize == "n" or inputVisualize !=
"N"):
                                print("Back to previous menu...")
                                print(
                                    "-------------------------------------")
                    elif (countPoint.isdigit() and int(countPoint) > 10000):
                        print("Too many points. Maximum points are 10000")
                    elif (countPoint.isdigit() and int(countPoint) <= 1):
                        print("Please input more than one points")
                    elif (not countPoint.isdigit()):
                        print("Please input positive integer only")

                elif (not r_n.isdigit() and r_n != "e"):
                    print("Please input positive integer only")

                elif (r_n != "e" and int(r_n) > 100):
                    print("Too many dimension. Keep it under 100")
                elif (r_n == "e"):
                    startPoint = False

                print("")
        elif (inputMenu == "2"):
            findFile = True
            while (findFile):
                try:
                    inputFileName = input(
                        "Please input filename or type 'e' to return to
previous menu: ")
                    if (inputFileName == "e"):
                        findFile = False
                    else:
                        try:
                            splitText, dimension, numberOfPoints =
ip.inputFile(
                                inputFileName)
                            _points = ip.processPoints(
                                splitText, dimension, numberOfPoints)
                            _points, _minDNC, resultDNC, calledDNC,
finalTimeDNC, _minBF, resultBF, calledBF, finalTimeBF = op.result(
                                _points)
```

```
                            op.printToTerminal(
                                _minDNC, resultDNC, calledDNC, finalTimeDNC,
_minBF, resultBF, calledBF, finalTimeBF)
                            toFile = input("Print to File? y/n ")

                            while toFile != "y" and toFile != "Y" and toFile !=
"n" and toFile != "N":
                                print("Please input between y or n")
                                print(
                                    "-------------------------------------")
                                toFile = input("Print to File? y/n ")
                            if (toFile == "y" or toFile == "Y"):
                                fileName = input("File Name: ")
                                op.outputToFile(fileName, dimension,
numberOfPoints, _minDNC,
                                                resultDNC, calledDNC,
finalTimeDNC, _minBF, resultBF, calledBF, finalTimeBF)

                            elif (toFile == "n" or toFile == "N"):
                                findFile = False

                            if(dimension == 3):
                                print("Do you want to visualize the data? y/n")
                                inputVisualize = input("Choice: ")
                                while inputVisualize != "y" and inputVisualize
!= "Y" and inputVisualize != "n" and inputVisualize != "N":
                                    print("Please input between y or n")
                                    print(

"-------------------------------------")
                                    inputVisualize = input("Choice: ")
                                if (inputVisualize == "y" or inputVisualize ==
"Y"):
                                    fileName = input("Figure Name: ")
                                    print("Visualizing...")
                                    vs.visualize(_points, resultDNC, fileName)
                                elif (inputVisualize == "n" or inputVisualize
== "N"):
                                    print("Back to previous menu...")
                                    print(

"-------------------------------------")
                        except Exception as err:
                            print(err)
                            print("")
                        except:
                            print("Incorrect File configuration")

                except FileNotFoundError:
                    print("File not found")

        elif (inputMenu == "3"):
            op.printGoodbye()
            start = False
```

```
        else:
            print("The option is between 1 or 2")
```

3. Contoh masukan dan luaran (skrinsut)

Program ini di uji pada platform:

'platform': Linux,

'platform-release': 5.15.79.1-microsoft-standard-WSL2,

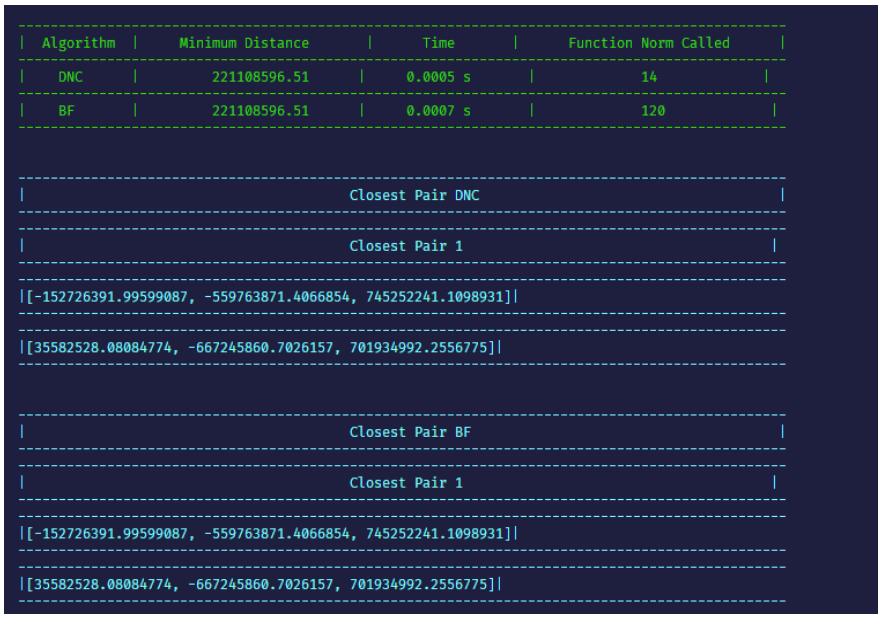
'platform-version': #1 SMP Wed Nov 23 01:01:46 UTC 2022,

'architecture': x86_64,

'processor': x86_64,

'ram': 10.37125632 GB

a. Titik berjumlah 16 pada dimensi 3 dengan hasil titik randomize



Algorithm	Minimum Distance	Time	Function Norm Called
DNC	221108596.51	0.0005 s	14
BF	221108596.51	0.0007 s	120

Closest Pair DNC

Closest Pair 1

[-152726391.99599087, -559763871.4066854, 745252241.1098931]

[35582528.08084774, -667245860.7026157, 701934992.2556775]

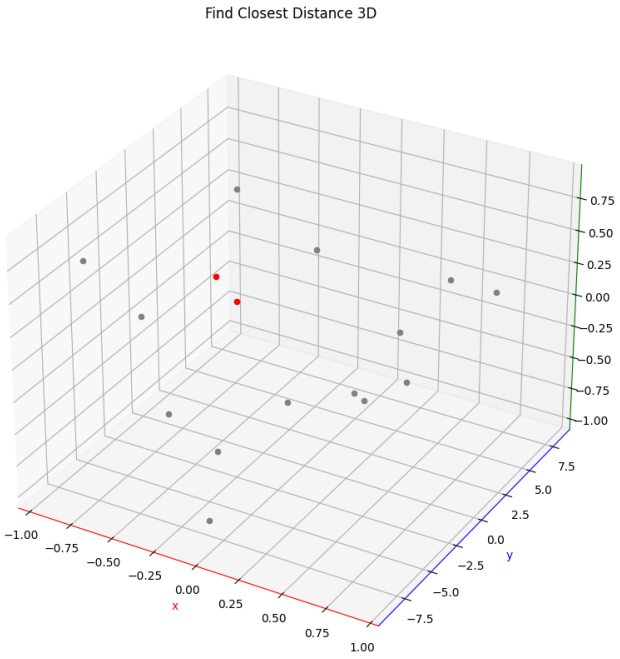
Closest Pair BF

Closest Pair 1

[-152726391.99599087, -559763871.4066854, 745252241.1098931]

[35582528.08084774, -667245860.7026157, 701934992.2556775]

Gambar 3.a.1 Hasil Eksekusi



Gambar 3.a.2 Hasil Plot

b. Titik berjumlah 64 pada dimensi 3 dengan hasil titik randomize

Algorithm	Minimum Distance	Time	Function Norm Called
DNC	69256424.34	0.0031 s	69
BF	69256424.34	0.0099 s	2016

Closest Pair DNC

Closest Pair 1

[762674617.1100693, -360960073.24838305, -871090544.1674552]

[806224588.593003, -379323312.9349245, -921713105.5901619]

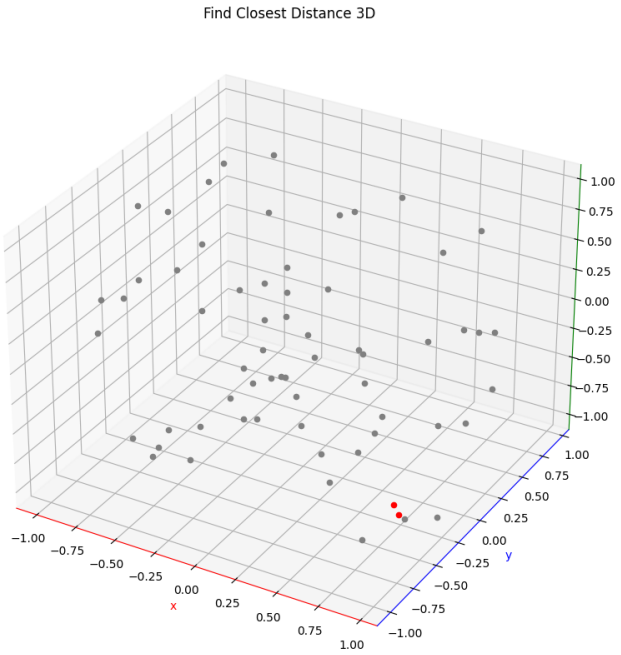
Closest Pair BF

Closest Pair 1

[762674617.1100693, -360960073.24838305, -871090544.1674552]

[806224588.593003, -379323312.9349245, -921713105.5901619]

Gambar 3.b.1 Hasil Eksekusi



Gambar 3.b.2 Hasil Plot

c. Titik berjumlah 128 pada dimensi 3 dengan hasil titik randomize

Algorithm	Minimum Distance	Time	Function Norm Called
DNC	18871267.33	0.0069 s	112
BF	18871267.33	0.1194 s	8128

Closest Pair DNC

Closest Pair 1

[-350844324.03153837, 465184368.8455248, 532467290.1388521]

[-345407047.84896946, 462253156.6746826, 514635611.767504]

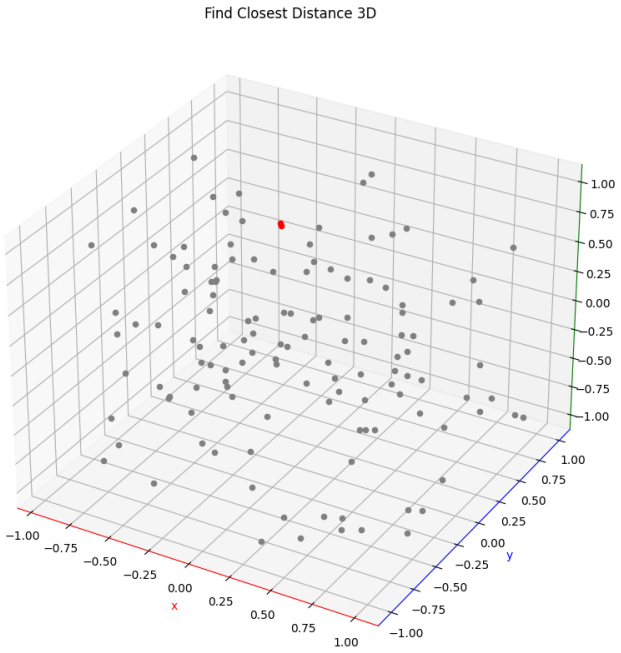
Closest Pair BF

Closest Pair 1

[-350844324.03153837, 465184368.8455248, 532467290.1388521]

[-345407047.84896946, 462253156.6746826, 514635611.767504]

Gambar 3.c.1 Hasil Eksekusi



Gambar 3.c.2 Hasil Plot

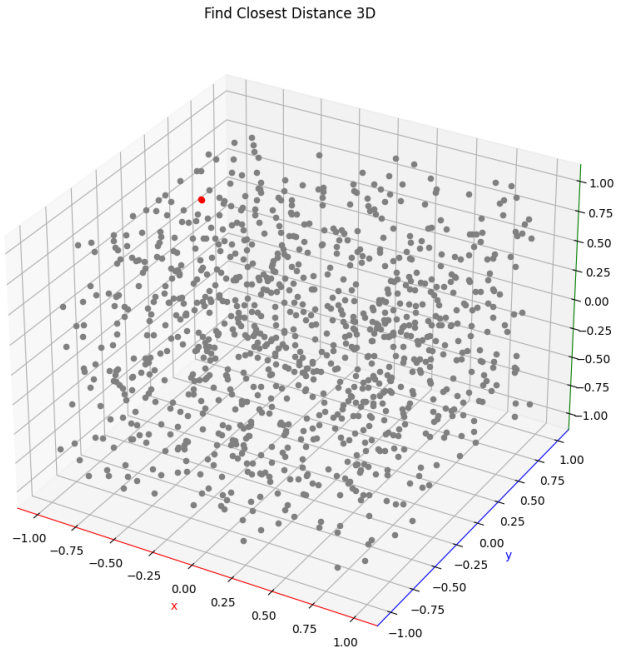
d. Titik berjumlah 1000 pada dimensi 3 dengan hasil titik randomize

Algorithm	Minimum Distance	Time	Function Norm Called
DNC	3937529.27	0.0427 s	984
BF	3937529.27	1.7692 s	499500

Closest Pair DNC
Closest Pair 1
[-729969339.4256197, 188873605.8274026, 769174720.0714564]
[-726578644.3996565, 186893380.2764287, 769468030.0375898]

Closest Pair BF
Closest Pair 1
[-729969339.4256197, 188873605.8274026, 769174720.0714564]
[-726578644.3996565, 186893380.2764287, 769468030.0375898]

Gambar 3.d.1 Hasil Eksekusi



Gambar 3.d.2 Hasil Plot

e. Titik berjumlah 16 pada dimensi 4 dengan hasil titik randomize

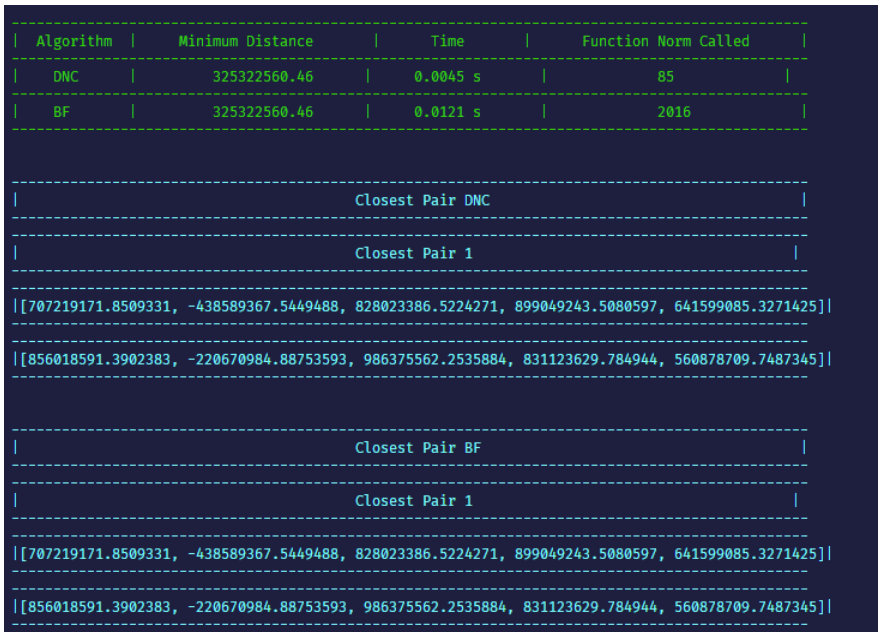
Algorithm	Minimum Distance	Time	Function Norm Called
DNC	335706510.18	0.0006 s	14
BF	335706510.18	0.0011 s	120

Closest Pair DNC
Closest Pair 1
[259302285.0394299, -539041303.5788406, 313699192.3911786, 693009396.3378453]
[355374167.2781582, -654728032.0591671, 519031707.32554173, 474093445.38225794]

Closest Pair BF
Closest Pair 1
[259302285.0394299, -539041303.5788406, 313699192.3911786, 693009396.3378453]
[355374167.2781582, -654728032.0591671, 519031707.32554173, 474093445.38225794]

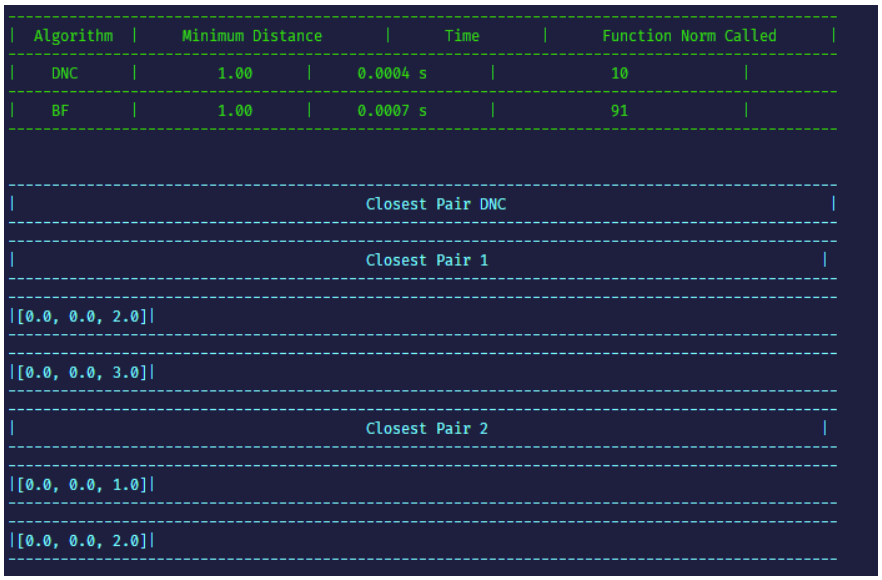
Gambar 3.e.1 Hasil Eksekusi

f. Titik berjumlah 64 pada dimensi 5 dengan hasil titik randomize



Gambar 3.f.1 Hasil Eksekusi

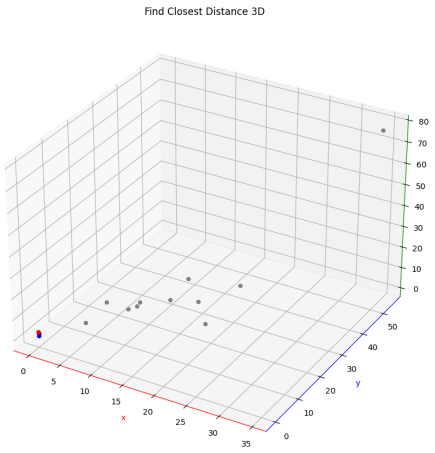
g. Titik berjumlah 16 pada dimensi 3 dengan input file



Gambar 3.g.1 Hasil Eksekusi

```
|                                    Closest Pair BF                                    |
|                                    Closest Pair 1                                     |
|[0.0, 0.0, 1.0]|
|[0.0, 0.0, 2.0]|
|                                    Closest Pair 2                                     |
|[0.0, 0.0, 2.0]|
|[0.0, 0.0, 3.0]|
```

Gambar 3.g.2 Hasil Eksekusi



Gambar 3.g.3 Hasil Plot

4. Pranala github

https://github.com/debbyalmadea/Tucil2_13521132_13521153

5. Checklist

Poin	Ya	Tidak
Program berhasil dikompilasi tanpa kesalahan	✓	
Program berhasil *running*	✓	
Program dapat membaca input/*generate* sendiri dan memberikan luaran	✓	

Luaran program sudah benar (solusi closest pair benar)	✓	
Bonus 1 dikerjakan	✓	
Bonus 2 dikerjakan	✓	