

**Tugas Kecil 2 IF2211 Strategi Algoritma**

**Semester II tahun 2022/2023**

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



Disusun oleh:

Muhammad Bangkit Dwi Cahyono 13521055

Mutawally Nawwar 13521065

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**2023**

## 1. Spesifikasi Tugas

Mencari sepasang titik terdekat dengan Algoritma Divide and Conquer sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Pada Tucil 2 kali ini Anda diminta 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:

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

Buatlah program dalam dalam Bahasa C/C++/Java/Python/Golang/Ruby/Perl (pilih salah satu) untuk mencari sepasang titik yang jaraknya terdekat satu sama lain dengan menerapkan algoritma divide and conquer untuk penyelesaiannya, dan perbandingannya dengan Algoritma Brute Force.

## 2. Penjelasan *Divide and Conquer*

*Divide and conquer* merupakan pendekatan masalah yang berprinsip memecah-mecah permasalahan yang terlalu besar menjadi beberapa bagian kecil sehingga lebih mudah untuk diselesaikan. Umumnya, strategi divide and conquer akan menyelesaikan permasalahan dengan membagi menjadi beberapa sub masalah serupa yang lebih kecil dari permasalahan awalnya. Kemudian akan dicari solusi-solusi dari sub masalah tersebut, ketika submasalah digabungkan akan bisa membuat solusi untuk permasalahan yang lebih besar.

*Divide and conquer* berkaitan erat dengan implementasi algoritma rekursif. Algoritma rekursif adalah algoritma yang terdiri atas dua bagian, yaitu *base case* dan *recurrence*. *Base case* dari suatu algoritma rekursif merupakan kondisi dimana rekursivitas atau perulangan algoritma tersebut berhenti. Biasanya, *base case* dari algoritma rekursif merupakan *initial value* atau kasus terkecil yang tidak dapat dibagi

lebih lanjut. Kemudian *recurrence* dari algoritma rekursif merupakan kondisi dimana terdapat pemanggilan kembali algoritma rekursif tersebut (pemanggilan dalam kasus ini dapat diartikan sebagai pemanggilan fungsi, prosedur, metode, ataupun ADT rekursif) dengan nilai masukkan yang mendekati base case.

Algoritma *Divide and Conquer* memiliki tiga bagian utama, yaitu:

a. *Divide*

Bagian ini merupakan bagian untuk membagi permasalahan yang besar menjadi permasalahan yang lebih kecil. Permasalahan yang lebih kecil ini harus identik atau cara penyelesaiannya sama dengan permasalahan besar. Hal ini yang menyebabkan solusi untuk masalah kecil dapat juga menjadi solusi untuk masalah yang lebih besar.

b. *Conquer*

Bagian ini merupakan bagian untuk menyelesaikan tiap sub masalah. Jika dalam sub persoalan sudah menjadi persoalan terkecil (*base case*), Maka dapat diselesaikan secara langsung. Namun, jika tidak akan menjalankan kembali bagian *divide*.

c. *Combine*

Bagian ini merupakan bagian untuk menentukan persoalan utama. Dari sub masalah yang telah didapat solusinya, bagian ini akan menentukan solusi mana yang dapat atau menggabungkan solusi dari setiap sub masalah.

Pada umumnya, algoritma *divide and conquer* memiliki kompleksitas yang lebih cepat dibanding dengan algoritma *brute force*. Sebagai contoh, *bubble sort* memiliki kompleksitas  $O(n^2)$  sedangkan *merge sort* memiliki kompleksitas  $O(n \log(n))$ . Kedua algoritma tersebut merupakan algoritma untuk mengurutkan *list* hanya saja *bubble sort* menggunakan pendekatan *brute force* dan *merge sort* menggunakan *divide and conquer*.

*Divide and conquer* dapat menyelesaikan berbagai masalah, salah satunya adalah pencarian jarak titik terpendek. Berikut adalah langkah-langkah penerapan *divide and conquer* pada pencarian jarak titik terendah.

a. *Divide*

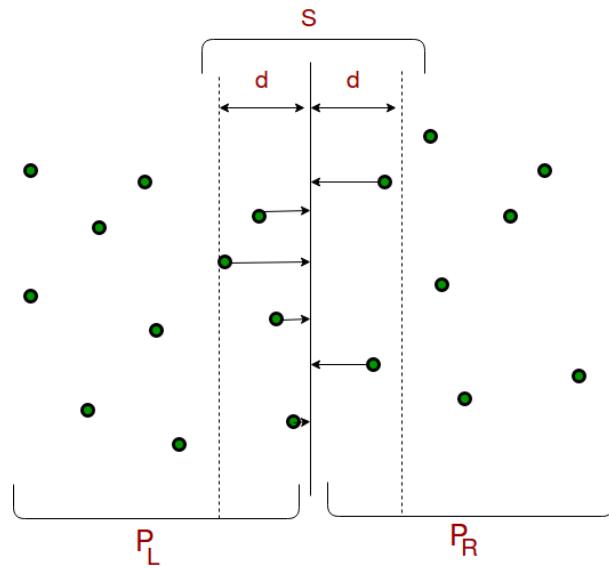
Pada bagian ini, sekumpulan titik akan dibagi menjadi 2 bagian, misalkan kanan dan kiri. Sub bagian tersebut akan dibagi secara terus menerus sampai setiap bagian hanya berisikan dua titik atau satu titik.

b. *Conquer*

Pada bagian ini, akan diselesaikan setiap sub bagianya. Ketika sub bagian ini hanya terdiri dari satu titik maka akan mengembalikan jarak terendahnya sebagai tak hingga. Hal ini dilakukan supaya jarak titik dengan dirinya sendiri tidak dianggap sebagai solusi. Ketika sub bagian hanya terdiri dari 2 titik, maka akan mengembalikan jarak dari dua titik tersebut karena sudah dipastikan itu merupakan solusi untuk sub bagian itu. Selain itu, akan dijalankan kembali bagian *divide*.

c. *Combine*

Pada bagian ini, setiap solusi dari bagian kanan dan kiri dibandingkan dan diambil yang jarak minimalnya paling kecil. Namun, hasil tersebut belum hasil mutlak karena kita harus meninjau titik-titik yang berada di area strip (tengah). Misalkan  $d$  adalah jarak minimum yang diperoleh dari  $\min(\text{kanan}, \text{kiri})$ , maka kita hanya mencarinya di area titik tengah  $+ d$  ke arah kanan dan titik tengah  $- d$ . Berikut ilustrasinya.



Proses ini ditingkatkan efisiensinya dengan mengabaikan kombinasi dimana jarak pada suatu dimensi lebih besar sama dengan dari jarak terkecil yang sudah didapat. Terlihat seperti  $O(n^2)$  tetapi ternyata hanya  $O(n)$  karena kita *break* atau skip iterasi dimana jaraknya lebih besar sama dengan dari jarak terkecil yang sudah didapat. Selanjutnya hasil minimum dari strip area dibandingkan dengan  $d$  yang sudah didapat sebelumnya. Program akan terus berjalan hingga didapatkan sepasang titik dan minimum jaraknya.

### 3. *Source Code* dengan bahasa Python

a. main.py

```

● ● ●

from type import Point, TwoPointDistance
from ioProcedure import inputProc, printResPoint
from algorithm import minimumTwoPointDistance, bruteForce
import algorithm

# plotting
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

```

```
# time
import time

# platform
import platform

n: int = 0
d: int = 0
listPoint: list[Point] = []
result: TwoPointDistance

print("====")
print("== CLOSEST POINT ==")
print("====")
print()

# Input banyak point (n) dan dimensi (d)
inputProc(n, d, listPoint)

if (listPoint[0].dimension < 2 or len(listPoint) < 2):
    print("Maaf, program hanya bisa menghitung point dengan dimensi > 1 dan titik > 2")
else:
    print()
    print("--- System Specs ---")
    my_system = platform.uname()
    print(f"System : {my_system.system}")
    print(f"Node Name : {my_system.node}")
    print(f"Release : {my_system.release}")
    print(f"Version : {my_system.version}")
    print(f"Machine : {my_system.machine}")
    print(f"Processor : {my_system.processor}")
    print()

    start_time = time.time()
    result, resultList = minimumTwoPointDistance(listPoint)
    res_time = time.time() - start_time
    print("--- Divide & Conquer Algorithm ---")
    print("Execution time: %s seconds" % (res_time))
    print("Count amount: ", algorithm.countDNC)
    print("Jarak terpendek yang didapat: ", result.distance, " satuan")

    printResPoint(resultList, result.index1)
    printResPoint(resultList, result.index2)
    print()

    start_time = time.time()
    resultBF = bruteForce(listPoint)
    res_time = time.time() - start_time
    print("--- Brute Force Algorithm ---")
    print("Execution time: %s seconds" % (res_time))
    print("Count amount: ", algorithm.countBF)
    print("Jarak terpendek yang didapat: ", resultBF.distance, " satuan")

    printResPoint(listPoint, resultBF.index1)
    printResPoint(listPoint, resultBF.index2)
    print()
```

```
if (listPoint[0].dimension == 3):
    fig = plt.figure()
    ax = plt.axes(projection='3d')

    x_values = []
    y_values = []
    z_values = []

    for i in range(len(resultList)):
        if (i == result.index1 or i == result.index2):
            xdata = resultList[i].values[0]
            x_values.append(xdata)

            ydata = resultList[i].values[1]
            y_values.append(ydata)

            zdata = resultList[i].values[2]
            z_values.append(zdata)

            ax.scatter3D(xdata, ydata, zdata, color = "red")
        else:
            xdata = resultList[i].values[0]
            ydata = resultList[i].values[1]
            zdata = resultList[i].values[2]

            ax.scatter3D(xdata, ydata, zdata, color =
"blue")
            ax.plot(x_values, y_values, z_values, color = "red")
            ax.set_title('3D Points', fontsize = 14)
            ax.set_xlabel('x')
            ax.set_ylabel('y')
            ax.set_zlabel('z')

            plt.show()
elif (listPoint[0].dimension == 2):
    fig = plt.figure()

    x_values = []
    y_values = []

    for i in range(len(resultList)):
        if (i == result.index1 or i == result.index2):
            xdata = resultList[i].values[0]
            x_values.append(xdata)
            ydata = resultList[i].values[1]
            y_values.append(ydata)
            plt.scatter(xdata, ydata, color = "red")
        else:
            xdata = resultList[i].values[0]
            ydata = resultList[i].values[1]
            plt.scatter(xdata, ydata, color = "blue")

    plt.plot(x_values, y_values, color = "red")
    plt.title('2D Points', fontsize = 14)
    plt.xlabel('x')
    plt.ylabel('y')

    plt.show()
```

b. type.py

```
● ● ●

class Point:
    def __init__(self, dimension: int, values: list[float]):
        self.dimension = dimension
        self.values = values

class TwoPointDistance:
    def __init__(self, distance: float, index1: int, index2: int):
        self.distance = distance
        self.index1 = index1
        self.index2 = index2
```

c. ioProcedure.py

```
● ● ●

from type import Point
import math
import random

def inputProc(n: int, d: int, listPoint: list[Point]):
    # n : banyak titik
    n = int(input("Masukan jumlah titik: "))

    # d : dimensi
    d = int(input("Masukan derajat titik: "))

    while (len(listPoint) < n):
        tmpPoint: Point
        tmpValues: list[float] = []
        for i in range(d):
            tmpValues.append(round(random.uniform(-1000, 1000), 2))
        tmpPoint = Point(d, tmpValues)

        # check agar tidak duplikat
        if (tmpPoint not in listPoint):
            listPoint.append(tmpPoint)

def printResPoint(listPoint: list[Point], index: int):
    print("<", end='')
    for i in range(listPoint[0].dimension):
        print(listPoint[index].values[i], end=' ')
        if (i != listPoint[0].dimension - 1):
            print(',', end=' ')
    print(">")
```

#### d. algorithm.py

```
● ● ●

from type import Point, TwoPointDistance
import math

countBF = 0
countDNC = 0

def bruteForce(listPoint: list[Point]) -> TwoPointDistance:
    global countBF
    # Algoritma Brute Force
    # mencari jarak terdekat dengan mencoba semua kemungkinan dari point yang ada
    if (len(listPoint) == 1):
        return TwoPointDistance(float('inf'), 0, 0)
    else:
        resMinDistance: float = TwoPointDistance(
            distance(listPoint[0], listPoint[1]), 0, 1)
        countBF += 1

        for i in range(len(listPoint)):
            for j in range(i + 1, len(listPoint)):
                if distance(listPoint[i], listPoint[j]) < resMinDistance.distance:
                    resMinDistance = TwoPointDistance(
                        distance(listPoint[i], listPoint[j]), i, j)
            countBF += 1

    return resMinDistance

def minimumTwoPointDistance(listPoint: list[Point]) -> (TwoPointDistance, list[Point]):
    # Menghasilkan TwoPointDistance yang berisi
    # distance, indeks Point 1, indeks Point 2

    # Pre-processing
    # List terurut berdasarkan x (dimensi pertama)
    mergeSortPoint(listPoint, 0)

    return (divide(listPoint, 0, len(listPoint)-1), listPoint)

def stripClosest(strip: list[Point], stripIndex: list[int], minDistance: TwoPointDistance) ->
    TwoPointDistance:
    # Menghasilkan TwoPointDistance pada Strip Area
    min_distance: float = minDistance.distance
    resMinDistance: TwoPointDistance = minDistance
    global countDNC

    # Pre-processing
    # List terurut berdasarkan y (dimensi kedua)
    combined = [list(a) for a in zip(strip, stripIndex)]
    mergeSortZip(combined, 1)

    stripIndex = [y for x, y in combined]
    strip = [x for x, y in combined]

    for i in range(len(strip)):
        for j in range(i + 1, len(strip)):
            # Jika saat perbandingan indeks dan indeks setelahnya sudah lebih dari
            # distance, maka break karena untuk indeks2 selanjutnya pasti > min_distance
            if (strip[j].values[1] - strip[i].values[1] >= min_distance):
                break

            # Jika lebih kecil, update
            temp_distance = distance(strip[i], strip[j])
            countDNC += 1
            if temp_distance < min_distance:
                min_distance = temp_distance
                resMinDistance = TwoPointDistance(
                    temp_distance, stripIndex[i], stripIndex[j])

    return resMinDistance
```

```

def divide(listPoint: list[Point], firstI: int, lastI: int) -> TwoPointDistance:
    # Kamus lokal
    tmpRes1: TwoPointDistance
    tmpRes2: TwoPointDistance
    resMinDistance: TwoPointDistance
    global countDNC

    # Jika first index = last index
    # Kasus jumlah elemen = 1
    if (firstI == lastI):
        return TwoPointDistance(float('inf'), firstI, firstI)
    # Kasus jumlah elemen = 2
    elif (firstI+1 == lastI):
        countDNC += 1
        return TwoPointDistance(distance(listPoint[firstI], listPoint[lastI]), firstI, lastI)
    # Kasus lain
    else:
        # Bagi 2 bagian left, right
        mid: int = (firstI+lastI)//2
        midPoint: Point = listPoint[mid]

        # RECURSIVE
        # tmpRes1 bagian kiri
        tmpRes1 = divide(listPoint, firstI, mid)
        # tmpRes2 bagian kanan
        tmpRes2 = divide(listPoint, mid+1, lastI)

        # Menentukan minimum distance
        if (tmpRes1.distance < tmpRes2.distance):
            resMinDistance = tmpRes1
        else:
            resMinDistance = tmpRes2

        # Melintasi kasus pada Strip Area
        strip: list[Point] = []
        stripIndex: list[int] = []
        for i in range(firstI, lastI + 1):
            if abs(listPoint[i].values[0] - midPoint.values[0]) < resMinDistance.distance:
                strip.append(listPoint[i])
                stripIndex.append(i)

        resStripDistance: TwoPointDistance = stripClosest(
            strip, stripIndex, resMinDistance)

        if (resMinDistance.distance < resStripDistance.distance):
            return resMinDistance
        else:
            return resStripDistance

def distance(point1: Point, point2: Point) -> float:
    sum: int = 0
    for i in range(point1.dimension):
        sum += (point1.values[i]-point2.values[i])**2
    return math.sqrt(sum)

def mergeSortPoint(listPoint: list[Point], index: int):
    if len(listPoint) > 1:
        mid: int
        i: int
        j: int
        k: int
        lengthLeft: int
        lengthRight: int
        leftHalf: list[Point]
        rightHalf: list[Point]
        # Bagi menjadi 2 bagian
        mid = len(listPoint) // 2
        leftHalf = listPoint[:mid]
        rightHalf = listPoint[mid:]

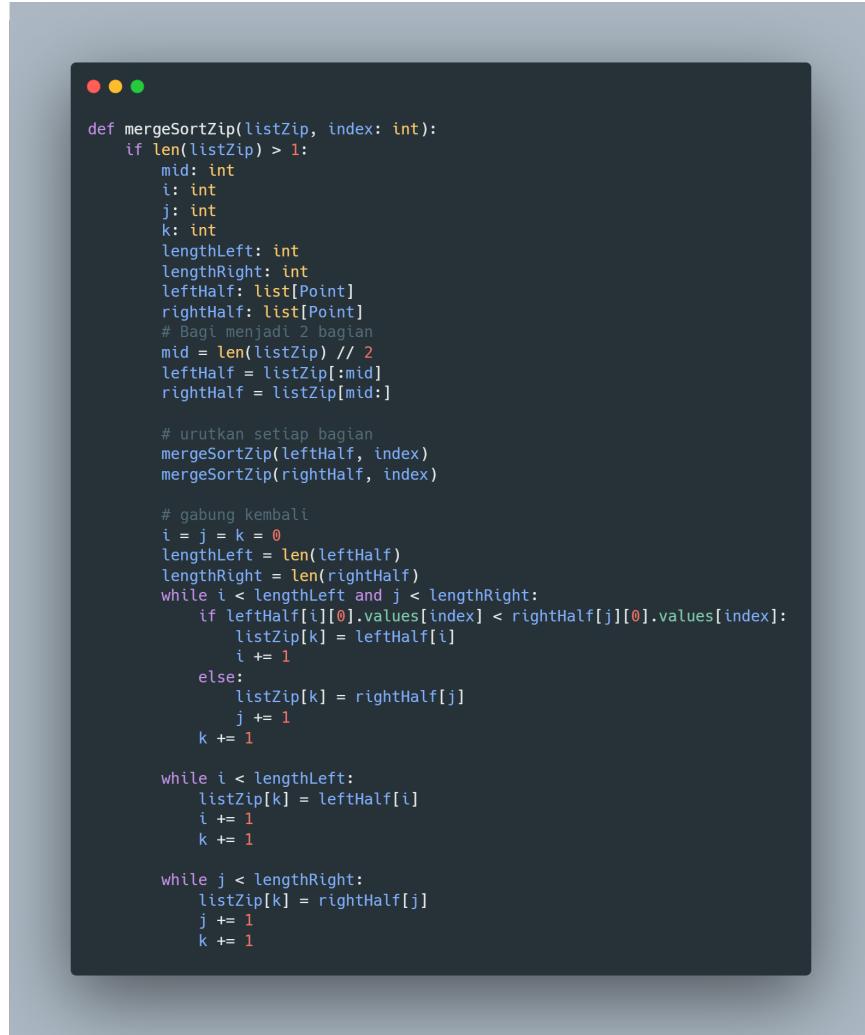
        # urutkan setiap bagian
        mergeSortPoint(leftHalf, index)
        mergeSortPoint(rightHalf, index)

        # gabung kembali
        i = j = k = 0
        lengthLeft = len(leftHalf)
        lengthRight = len(rightHalf)
        while i < lengthLeft and j < lengthRight:
            if getattr(leftHalf[i], "values")[index] < getattr(rightHalf[j], "values")[index]:
                listPoint[k] = leftHalf[i]
                i += 1
            else:
                listPoint[k] = rightHalf[j]
                j += 1
            k += 1

        while i < lengthLeft:
            listPoint[k] = leftHalf[i]
            i += 1
            k += 1

        while j < lengthRight:
            listPoint[k] = rightHalf[j]
            j += 1
            k += 1

```



```
def mergeSortZip(listZip, index: int):
    if len(listZip) > 1:
        mid: int
        i: int
        j: int
        k: int
        lengthLeft: int
        lengthRight: int
        leftHalf: list[Point]
        rightHalf: list[Point]
        # Bagi menjadi 2 bagian
        mid = len(listZip) // 2
        leftHalf = listZip[:mid]
        rightHalf = listZip[mid:]

        # urutkan setiap bagian
        mergeSortZip(leftHalf, index)
        mergeSortZip(rightHalf, index)

        # gabung kembali
        i = j = k = 0
        lengthLeft = len(leftHalf)
        lengthRight = len(rightHalf)
        while i < lengthLeft and j < lengthRight:
            if leftHalf[i][0].values[index] < rightHalf[j][0].values[index]:
                listZip[k] = leftHalf[i]
                i += 1
            else:
                listZip[k] = rightHalf[j]
                j += 1
            k += 1

        while i < lengthLeft:
            listZip[k] = leftHalf[i]
            i += 1
            k += 1

        while j < lengthRight:
            listZip[k] = rightHalf[j]
            j += 1
            k += 1
```

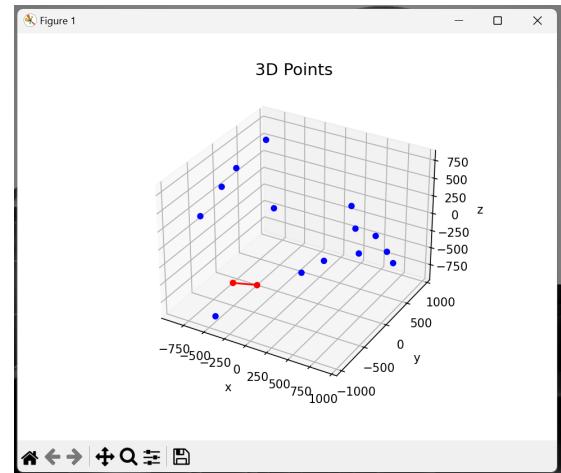
## 4. Pengujian

### a. Test case 1

input	$n = 16$ dimensi = 3
-------	-------------------------

output

```
===== CLOSEST POINT =====  
=====  
Masukan jumlah titik: 16  
Masukan derajat titik: 3  
--- System Specs ---  
System : Windows  
Node Name : bangkitdc  
Release : 10  
Version : 10.0.22621  
Machine : AMD64  
Processor : AMD64 Family 25 Model 68 Stepping 1, AuthenticAMD  
--- Divide & Conquer Algorithm ---  
Execution time: 0.0 seconds  
Count amount: 25  
Jarak terpendek yang didapat: 281.07015352043345 satuan  
<-605.7,-337.82,-778.52>  
<-331.52,-297.24,-731.84>  
--- Brute Force Algorithm ---  
Execution time: 0.0 seconds  
Count amount: 121  
Jarak terpendek yang didapat: 281.07015352043345 satuan  
<-605.7,-337.82,-778.52>  
<-331.52,-297.24,-731.84>
```



b. Test case 2

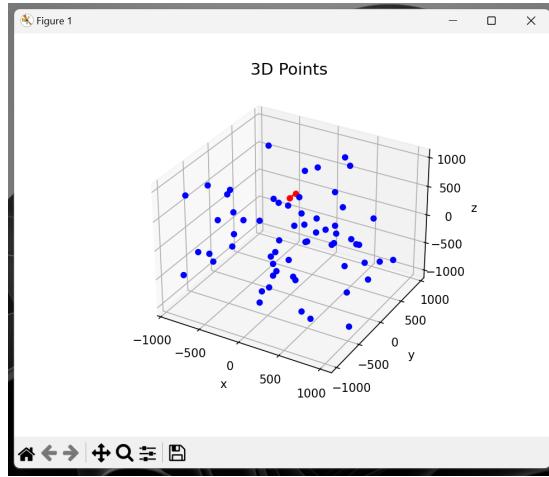
input

$n = 64$

dimensi = 3

output

```
=====  
===== CLOSEST POINT =====  
=====  
  
Masukan jumlah titik: 64  
Masukan derajat titik: 3  
  
--- System Specs ---  
System : Windows  
Node Name : bangkitdc  
Release : 10  
Version : 10.0.22621  
Machine : AMD64  
Processor : AMD64 Family 25 Model 68 Stepping 1, AuthenticAMD  
  
--- Divide & Conquer Algorithm ---  
Execution time: 0.0009915828704833984 seconds  
Count amount: 197  
Jarak terpendek yang didapat: 98.40370369046084 satuan  
<-497.72, 844.68, -175.02>  
<-452.44, 908.44, -115.29>  
  
--- Brute Force Algorithm ---  
Execution time: 0.0009992122650146484 seconds  
Count amount: 2017  
Jarak terpendek yang didapat: 98.40370369046084 satuan  
<-497.72, 844.68, -175.02>  
<-452.44, 908.44, -115.29>
```



c. Test case 3

input

$n = 128$   
dimensi = 3

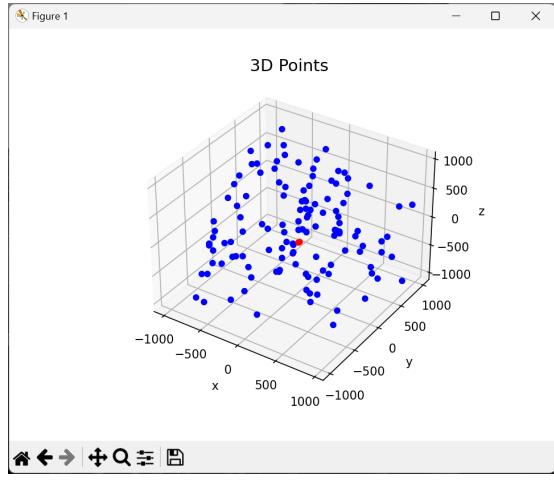
output

```
===== CLOSEST POINT =====
=====
Masukan jumlah titik: 128
Masukan derajat titik: 3

--- System Specs ---
System      : Windows
Node Name   : bangkitdc
Release     : 10
Version     : 10.0.22621
Machine     : AMD64
Processor   : AMD64 Family 25 Model 68 Stepping 1, AuthenticAMD

--- Divide & Conquer Algorithm ---
Execution time: 0.001509428024291992 seconds
Count amount: 288
Jarak terpendek yang didapat: 23.196340228579164 satuan
<429.94,-584.23,457.75>
<442.21,-583.45,477.42>

--- Brute Force Algorithm ---
Execution time: 0.0050067901611328125 seconds
Count amount: 8129
Jarak terpendek yang didapat: 23.196340228579164 satuan
<429.94,-584.23,457.75>
<442.21,-583.45,477.42>
```



d. Test case 4

input

$n = 1000$

dimensi = 3

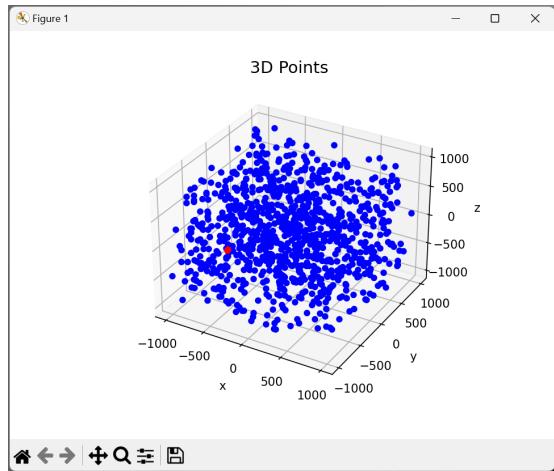
output

```
===== CLOSEST POINT =====
Masukan jumlah titik: 1000
Masukan derajat titik: 3

--- System Specs ---
System      : Windows
Node Name   : bangkitdc
Release     : 10
Version     : 10.0.22621
Machine     : AMD64
Processor   : AMD64 Family 25 Model 68 Stepping 1, AuthenticAMD

--- Divide & Conquer Algorithm ---
Execution time: 0.011005878448486328 seconds
Count amount: 4484
Jarak terpendek yang didapat: 13.841000686366616 satuan
<-304.92,-919.34,204.76>
<-305.73,-908.4,213.2>

--- Brute Force Algorithm ---
Execution time: 0.3215618133544922 seconds
Count amount: 499501
Jarak terpendek yang didapat: 13.841000686366616 satuan
<-305.73,-908.4,213.2>
<-304.92,-919.34,204.76>
```



e. Test case 5

input

$n = 100$

dimensi = 2

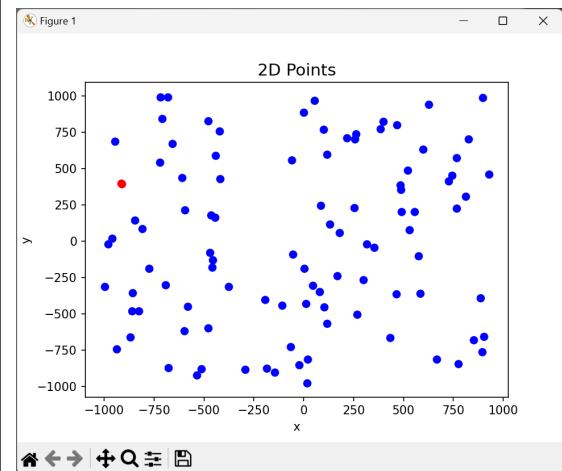
output

```
===== CLOSEST POINT =====
=====
Masukan jumlah titik: 100
Masukan derajat titik: 2

--- System Specs ---
System      : Windows
Node Name   : bangkitdc
Release     : 10
Version     : 10.0.22621
Machine     : AMD64
Processor   : AMD64 Family 25 Model 68 Stepping 1, AuthenticAMD

--- Divide & Conquer Algorithm ---
Execution time: 0.0009987354278564453 seconds
Count amount: 111
Jarak terpendek yang didapat: 5.986868964659207 satuan
<-911.47, 393.35>
<-914.98, 398.2>

--- Brute Force Algorithm ---
Execution time: 0.0019986629486083984 seconds
Count amount: 4951
Jarak terpendek yang didapat: 5.986868964659207 satuan
<-914.98, 398.2>
<-911.47, 393.35>
```



f. Test case 6

input

$n = 1000$

dimensi = 5

output

```
===== CLOSEST POINT =====  
=====  
Masukan jumlah titik: 1000  
Masukan derajat titik: 5  
--- System Specs ---  
System : Windows  
Node Name : bangkitdc  
Release : 10  
Version : 10.0.22621  
Machine : AMD64  
Processor : AMD64 Family 25 Model 68 Stepping 1, AuthenticAMD  
--- Divide & Conquer Algorithm ---  
Execution time: 0.034523963928222656 seconds  
Count amount: 27924  
Jarak terpendek yang didapat: 49.92906768606843 satuan  
<-665.08,355.71,-459.29,535.9,9.24>  
<-700.19,379.5,-467.73,515.92,-5.72>  
--- Brute Force Algorithm ---  
Execution time: 0.4420647621154785 seconds  
Count amount: 499501  
Jarak terpendek yang didapat: 49.92906768606843 satuan  
<-700.19,379.5,-467.73,515.92,-5.72>  
<-665.08,355.71,-459.29,535.9,9.24>
```

g. Test case 7

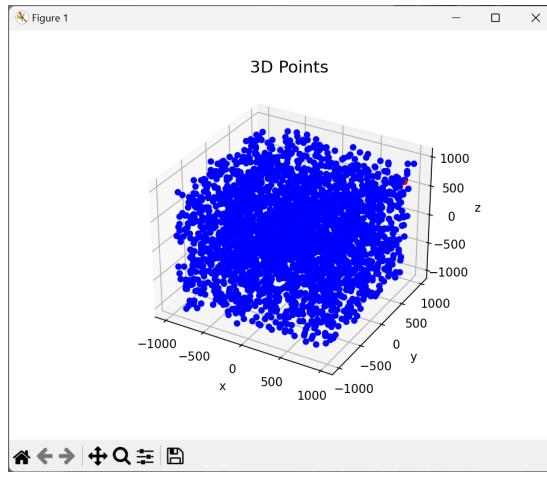
input	n = 1000  dimensi = 10
output	===== CLOSEST POINT ===== ===== Masukan jumlah titik: 1000 Masukan derajat titik: 10 --- System Specs --- System : Windows Node Name : bangkitdc Release : 10 Version : 10.0.22621 Machine : AMD64 Processor : AMD64 Family 25 Model 68 Stepping 1, AuthenticAMD --- Divide & Conquer Algorithm --- Execution time: 0.4796271324157715 seconds Count amount: 288746 Jarak terpendek yang didapat: 466.5375853026206 satuan <-258.62,-330.98,559.33,-776.61,281.55,927.67,916.38,-631.31,-75.15,-381.83> <-228.37,-248.5,576.46,-851.72,137.82,640.98,628.65,-555.06,10.36,-309.44> --- Brute Force Algorithm --- Execution time: 0.7661476135253906 seconds Count amount: 499501 Jarak terpendek yang didapat: 466.5375853026206 satuan <-258.62,-330.98,559.33,-776.61,281.55,927.67,916.38,-631.31,-75.15,-381.83> <-228.37,-248.5,576.46,-851.72,137.82,640.98,628.65,-555.06,10.36,-309.44>

h. Test case 8

input	n = 3000  dimensi = 3
-------	-----------------------------

output

```
===== CLOSEST POINT =====  
  
Masukan jumlah titik: 3000  
Masukan derajat titik: 3  
  
--- System Specs ---  
System : Windows  
Node Name : bangkitdc  
Release : 10  
Version : 10.0.22621  
Machine : AMD64  
Processor : AMD64 Family 25 Model 68 Stepping 1, AuthenticAMD  
  
--- Divide & Conquer Algorithm ---  
Execution time: 0.04151558876037598 seconds  
Count amount: 15498  
Jarak terpendek yang didapat: 6.256908182161502 satuan  
<950.34, 838.17, 696.24>  
<954.99, 841.17, 693.32>  
  
--- Brute Force Algorithm ---  
Execution time: 2.885156869883057 seconds  
Count amount: 4498501  
Jarak terpendek yang didapat: 6.256908182161502 satuan  
<950.34, 838.17, 696.24>  
<954.99, 841.17, 693.32>
```



## 5. Pranala *Google Drive* dan *Github*

### a. *Google Drive*

<https://drive.google.com/drive/folders/1MUI4BViSx3FzsOL6QR9U7026zlRLL-Az/>

### b. *Github*

[https://github.com/mutawalle/Tucil2\\_13521055\\_13521065.git](https://github.com/mutawalle/Tucil2_13521055_13521065.git)

## 6. Tabel Checklist

Poin	Ya	Tidak
1. Program berhasil dikompilasi 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. Mengerjakan Bonus 1	√	
6. Mengerjakan Bonus 2	√	

## 7. 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)
- [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf)
- [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian3.pdf)
- [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Divide-and-Conquer-\(2022\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Divide-and-Conquer-(2022)-Bagian4.pdf)
- <https://www.neliti.com/publications/271489/penerapan-algoritma-divide-and-conquer-pada-perancangan-sistem-identitas-pendudu>
- <https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/>