

Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

Semester II Tahun 2022/2023

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



Disusun Oleh :

Johann Christian Kandani (13521138)

Johanes Lee (13521148)

Kelas 2

**Program Studi S1 Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung**

2023

Bab I

Deskripsi Persoalan

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

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

Bab II

Algoritma Penyelesaian

Berikut merupakan penyelesaian persoalan pencarian dua titik terdekat yang sudah dideskripsikan dengan pendekatan *divide and conquer* (asumsi banyak titik di dalam *list* selalu besar dari 1 dan setiap titik sudah terurut menaik berdasarkan nilai elemen/dimensi pertama). Pada algoritma ini, penyelesaian digeneralisasikan untuk titik pada M dimensi.

1. Jika banyak titik lebih kecil atau sama dengan 3, selesaikan dengan pendekatan *Brute Force*:

1.1 Inisialisasi nilai jarak terkecil yang pernah diperoleh dengan bilangan yang sangat besar.

1.2 Untuk setiap N-1 titik pertama dengan N merupakan banyak titik pada *list*, lakukan hal berikut.

1.2.1 Untuk setiap titik dari indeks tepat di atas titik yang sudah dipilih ($i+1$ untuk i merupakan indeks titik yang dipilih pada langkah 1.2) hingga indeks terakhir *list*, lakukan hal berikut.

1.2.1.1 Hitung jarak Euclidean kedua titik yang telah dipilih pada M dimensi dengan rumus berikut.

$$\begin{aligned}d(\mathbf{p}, \mathbf{q}) &= d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} \\ &= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}.\end{aligned}$$

(Sumber: https://zims-en.kiwix.campusafrika.gos.orange.com/wikipedia_en_all_nopic/A/Euclidean_distance)

1.2.1.2 Jika jarak yang diperoleh lebih kecil daripada jarak terkecil yang pernah diperoleh sebelumnya, ganti jarak terkecil tersebut dengan jarak saat ini dan catat pasangan titik saat ini.

1.3 Kembalikan pasangan titik dengan jarak terkecil yang diperoleh sebagai hasil.

2. Selain itu (jika banyak titik dalam *list* lebih besar dari 3), lakukan pendekatan *divide and conquer* sebagai berikut.

2.1 *Divide*: bagi himpunan titik ke dalam dua bagian S_1 dan S_2 dengan S_1 mengandung $\lfloor N / 2 \rfloor$ titik pertama sedangkan S_2 mengandung semua titik sisanya.

- 2.2 *Conquer*: secara rekursif, terapkan seluruh algoritma dari langkah 1 untuk mencari pasangan titik terdekat pada S_1 dan S_2 .
- 2.3 *Combine*: terdapat tiga kemungkinan letak pasangan titik terdekat yang seharusnya menjadi solusi persoalan, yaitu di S_1 , S_2 , atau pasangan titik dipisahkan garis pembatas S_1 dan S_2 . Untuk mendapatkan solusi sebenarnya, lakukan hal berikut.
 - 2.3.1 Ambil pasangan titik dengan jarak minimum antara pasangan titik terdekat yang berada di dalam S_1 dengan pasangan titik terdekat yang berada di dalam S_2 sebagai solusi sementara, dengan jarak minimum tersebut adalah $temp$.
 - 2.3.2 Catat titik yang berada pada indeks $\lfloor N / 2 \rfloor$ sebagai titik yang berada di tengah *list* awal (merupakan anggota S_2), misalnya P_{mid} .
 - 2.3.3 Apabila titik-titik berada dalam 1 dimensi, lakukan hal berikut.
 - 2.3.3.1 Hitung selisih nilai elemen P_{mid} dengan elemen titik yang berada tepat di kirinya pada *list* tersebut (indeks $\lfloor N / 2 \rfloor - 1$).
 - 2.3.3.2 Jika selisih (jarak) yang diperoleh lebih kecil daripada $temp$, kembalikan pasangan titik ini sebagai solusi akhir. Jika tidak, kembalikan solusi sementara tadi sebagai solusi akhir.
 - 2.3.4 Selain itu, apabila titik-titik berada dalam 2 dimensi atau lebih, lakukan hal berikut.
 - 2.3.4.1 Untuk setiap titik pada *list*, hitung selisih elemen pada dimensi pertama titik tersebut dengan P_{mid} dan masukkan ke himpunan S_{Strip} jika selisihnya lebih kecil daripada $temp$.
 - 2.3.4.2 Urutkan setiap titik pada S_{Strip} sehingga terurut berdasarkan elemen kedua (dimensi kedua) setiap titik.
 - 2.3.4.3 Lakukan hal serupa dengan pendekatan *Brute Force* untuk titik-titik di dalam S_{Strip} , kecuali setiap pemilihan titik terurut berdasarkan urutan pada langkah 2.3.4.2 serta inisialisasi catatan jarak terdekat bernilai $temp$. Titik pertama yang sedang dipilih tidak perlu dibandingkan lagi dengan semua titik lainnya yang berada di kanan *list* S_{Strip} jika selisih elemen kedua (dimensi kedua) dari kedua titik tersebut lebih besar dari catatan jarak terdekat keseluruhan saat pencarian ini. Selain itu, kedua titik hanya perlu dicari jaraknya jika selisih setiap elemen kedua titik tersebut lebih kecil dari jarak terdekat yang tercatat saat ini.

2.3.4.4 Jika jarak terdekat yang diperoleh pada langkah 2.3.4.3 lebih kecil daripada *temp*, kembalikan pasangan titik dengan jarak terdekat yang baru tersebut sebagai solusi akhir. Jika tidak, kembalikan solusi sementara tadi sebagai solusi akhir.

Bab III

Implementasi Program

Berikut adalah implementasi program untuk mencari solusi persoalan dua titik terdekat yang digeneralisasikan pada persoalan N dimensi dalam bahasa Python menggunakan algoritma *divide and conquer*.

3.1 Points.py

```
import math
from numpy import random

class Counter:
    def __init__(self):
        self.count = 0

    def reset(self):
        self.count = 0

    def inc(self):
        self.count += 1

counter = Counter()

def getDistance(point1, point2):
    # mengembalikan jarak euclidean dua titik N dimensi
    result = 0
    counter.inc()

    for i in range(len(point1)):
        result += pow ((point1[i] - point2[i]), 2)

    return math.sqrt(result)

def isSmaller(point1, point2, startingIndex = 0):
    # membandingkan point1 dan point2, mengembalikan true jika ada indeks
    # terkecil sehingga elemen pertama point1 lebih kecil dari elemen point2
    # dengan indeks terkecil >= startingIndex

    # asumsi startingIndex < len(point1)
    for i in range(startingIndex, len(point1)):

        if (point1[i] < point2[i]):
            return True

        if (point1[i] > point2[i]):
            return False

    return False

def isInRange(point1, point2, minDistance):
    # mengembalikan true jika selisih tiap elemen kedua titik kecil dari
    minDistance

    for i in range(len(point1)):
        if (abs(point1[i] - point2[i]) >= minDistance):
            return False

    return True
```

```

def generateRandomPoints(length, dimension = 3, lowerBound = -1e6, upperBound = 1e6):
    points = [random.uniform(low=lowerBound, high=upperBound,
size=(dimension)).tolist() for i in range(length)]

    return points

def merge(points, left, mid, right, startingElementIndex):
    # menggabungkan dua list titik N dimensi yang sudah terurut
    # startingElementIndex menjadi acuan perbandingan dua buah titik
    leftLimit = mid - left + 1
    rightLimit = right - mid

    leftPoints = [points[i] for i in range(left, mid + 1)]
    rightPoints = [points[i] for i in range(mid + 1, right + 1)]

    i = 0
    j = 0
    k = left

    while (i < leftLimit and j < rightLimit):
        if (isSmaller(leftPoints[i], rightPoints[j], startingElementIndex)):
            points[k] = leftPoints[i]
            i += 1
        else:
            points[k] = rightPoints[j]
            j += 1

        k += 1

    while (i < leftLimit):
        points[k] = leftPoints[i]
        i += 1
        k += 1

    while (j < rightLimit):
        points[k] = rightPoints[j]
        j += 1
        k += 1

def mergeSort(points, left, right, startingElementIndex):
    # mengurutkan list titik dengan algoritma merge sort

    if (left >= right):
        return

    mid = (left + right) // 2

    mergeSort(points, left, mid, startingElementIndex)
    mergeSort(points, mid + 1, right, startingElementIndex)
    merge(points, left, mid, right, startingElementIndex)

def sort(points, startingElementIndex = 0):
    # mengurutkan list of points
    mergeSort(points, 0, len(points) - 1, startingElementIndex)

```

3.2 ClosestPair.py

```

import Points as p

```

```

def findNearestPairBF(points):
    # mengembalikan tuple berupa (minDistance, Pair) untuk pasangan titik
    terdekat dengan pendekatan brute force

    pair = []
    minDistance = float("inf")

    for i in range(len(points) - 1):
        for j in range(i + 1, len(points)):

            temp = p.getDistance(points[i], points[j])

            if (temp < minDistance):
                minDistance = temp
                pair = [points[i], points[j]]

    return (minDistance, pair)

def findNearestMid(points, mid, minDistance):
    # mengembalikan tuple berupa (minDistance, Pair) untuk pasangan titik
    terdekat yang berada di range tengah

    # kasus 1 dimensi

    if (len(points[0]) == 1):
        leftValue = (minDistance, [])

        if (mid > 0):
            leftValue = (abs(points[mid][0] - points[mid-1][0]), points[mid-1])

        return leftValue

    midPoint = points[mid]
    midPoints = []

    result = (minDistance, [])

    for point in points:
        if (abs(point[0] - midPoint[0]) < minDistance):
            midPoints.append(point)

    p.sort(midPoints, 1)

    size = len(midPoints)
    for i in range(size - 1):
        for j in range(i + 1, size):

            if (abs(midPoints[i][1] - midPoints[j][1]) >= result[0]):
                break

            if (not(p.isInRange(midPoints[i], midPoints[j], result[0]))):
                continue

            tempDistance = p.getDistance(midPoints[i], midPoints[j])

            if (tempDistance < result[0]):
                result = (tempDistance, [midPoints[i], midPoints[j]])

    return result

def findNearestPairDNC(points):
    # mengembalikan tuple berupa (minDistance, Pair) untuk pasangan titik
    terdekat dengan pendekatan divide and conquer
    if (len(points) <= 3):

```



```

print("Input format:")

if (mode == 3):
    print("N D                                {N is the number of points, and D is
the dimension of the points}")

else:
    print("N                                {N is the number of points}")
    print("D                                {D is the dimension of the points}")

if (mode != 2):
    print("\nPoints input must follow the format:")
    print("X1_1 X1_2 X1_3 ... X1_D")
    print("X2_1 X2_2 X2_3 ... X2_D")
    print("...")
    print("XN_1 XN_2 XN_3 ... XN_D          {where Xi_j represents the value of
i-th point in j-th dimension}")

print("-----\n")
# end function

def getChoices(choices, label = 'option', cancelOpt = False):
    print("\nSelect", label, ": ")

    optRange = len(choices)
    for i in range(optRange):
        print('[' + str(i+1) + ']', choices[i])

    # include additional option to return to previous menu
    if(cancelOpt):
        optRange += 1
        print(f'[{optRange}] Return')

    print()

    while(True):
        print('Your input : ', end='')
        userOpt = intInput(singleVal=True)

        if(userOpt != None):
            if(1 <= userOpt and userOpt <= optRange):
                break
            else:
                print(f'Input must be in range of [1-{optRange}]\n')
                # optRange = 1 not handled, why even use when only 1 option is
available?

        return userOpt
    # end function

def initPointsInput():
    while(True):
        print("Input number of points: ", end='')
        n = intInput(minRange=2, singleVal=True)
        if(n != None):
            break

    while(True):
        print("Input dimensions: ", end='')
        dimension = intInput(minRange=1, singleVal=True)
        if(dimension != None):
            break

    return n, dimension
# end function

```

```

def fileToPoints():
    filename = fixFileFormat(input("Input file path (relative to test folder):
"))

    try:
        f = open("./test/" + filename, "r")
    except FileNotFoundError:
        print("File not found.")
        return None

    tempIn = f.readline()
    pointsInfo = intInput(parseIn=tempIn, n=2)

    if(pointsInfo == None):
        print("File format does not follow input format.")
        f.close()
        return None
    else:
        # Validate number of points
        n = pointsInfo[0]
        if(n < 2):
            print("Number of points must be >= 2.")
            f.close()
            return None

        # Validate dimension value
        dim = pointsInfo[1]
        if(dim < 1):
            print("Points dimension cannot be lower than 1.")
            f.close()
            return None

    points = [None for _ in range(n)]

    for i in range(n):
        tempIn = f.readline()
        tempVal = realInput(parseIn=tempIn, n=dim)
        if(tempVal == None):
            print(f"invalid point value input at line {i+2}.")
            f.close()
            return None
        else:
            points[i] = tempVal

    f.close()
    return n, dim, points
# end function

def intInput(parseIn = None, n = 1, minRange = None, maxRange = None, singleVal =
False):
    # check parse parameter
    if (isinstance(parseIn, str)):
        parseList = list(parseIn.split())

    elif (isinstance(parseIn, list)):
        parseList = parseIn

    elif (parseIn == None):
        parseList = input().split()

    else:
        return None

    # validate input amount
    if(len(parseList) != n):
        print(f"Input must consist(s) of {n} integers!")
        return None

```

```

else:
    # validate input(s) type
    for i in range(0, n):
        try:
            parseList[i] = int(parseList[i])
        except ValueError:
            print(f"Input type must only be integers! (Caught invalid input
at position {i+1})")
            return None

    # Validate value(s) range if specified
    if(minRange != None): # validate min value
        for num in parseList:
            if(num < minRange):
                print(f"Input value(s) must be >= {minRange}!")
                return None

    if(maxRange != None): # validate max value
        for num in parseList:
            if(num > maxRange):
                print(f"Input value(s) must be <= {minRange}!")
                return None

    # if all input(s) valid, return list of numbers
    if(singleVal):
        return parseList[0]
    else:
        return parseList
# end function

def realInput(parseIn = None, n = 1, minRange = None, maxRange = None, singleVal
= False):
    # check parse parameter
    if (isinstance(parseIn, str)):
        parseList = list(parseIn.split())

    elif (isinstance(parseIn, list)):
        parseList = parseIn

    elif (parseIn == None):
        parseList = input().split()

    else:
        return None

    # validate input amount
    if(len(parseList) != n):
        print(f"Input must consist(s) of {n} real numbers!")
        return None
    else:
        # validate input(s) type
        for i in range(0, n):
            try:
                parseList[i] = float(parseList[i])
            except ValueError:
                print(f"Input type must only be real numbers! (Caught invalid
input at position {i+1})")
                return None

        # Validate value(s) range if specified
        if(minRange != None): # validate min value
            for num in parseList:
                if(num < minRange):
                    print(f"Input value(s) must be >= {minRange}!")
                    return None

        if(maxRange != None): # validate max value

```

```

        for num in parseList:
            if(num > maxRange):
                print(f"Input value(s) must be <= {minRange}!")
                return None

        # if all input(s) valid, return list of numbers
        if(singleVal):
            return parseList[0]
        else:
            return parseList
# end function

def plot3DPoints(Points, Point1, Point2):
    fig = plt.figure()
    ax = fig.add_subplot(projection = '3d')

    for point in Points:
        if(point == Point1):
            symbol = '^'
            color = '#ff0000'
        elif(point == Point2):
            symbol = '^'
            color = '#ffa500'
        else:
            symbol = 'o'
            color = '#7fdcf9'
        ax.scatter(point[0], point[1], point[2], marker=symbol, color=color)

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    print("\n[Waiting the plot to be closed]")
    plt.show()
# end function

def plot2DPoints(Points, Point1, Point2):
    # fig = plt.figure()
    # ax = fig.add_subplot(projection = '2d')

    for point in Points:
        if(point == Point1):
            symbol = '^'
            color = '#ff0000'
        elif(point == Point2):
            symbol = '^'
            color = '#ffa500'
        else:
            symbol = 'o'
            color = '#7fdcf9'
        plt.scatter(point[0], point[1], marker=symbol, color=color)

    plt.xlabel('X')
    plt.ylabel('Y')
    print("\n[Waiting the plot to be closed]\n")
    plt.show()
# end function

def fixFileFormat(fileName):

    return fileName.split('.', 1)[0] + ".txt"

```

3.3 main.py

```

import Points as p
import IO
import ClosestPair as cp
import time

# Main begin
IO.art()
IO.welcome()

while(True):
    run = IO.getChoices(['Start', 'Exit'], 'action')

    if (run == 1):
        # # boolean flag to mark input failure
        # failFlag = False

        choice = IO.getChoices(['Console', 'Random', 'File'], 'input mode',
cancelOpt=True)

        if(choice == 4): # user chose return
            continue

        IO.inputFormatInfo(choice)

        if (choice == 1):
            n, dim = IO.initPointsInput()
            points = [None for _ in range(n)]
            i = 0

            print("Input Points")
            while(i < n):

                while(True):
                    print("Point", i+1, ": ", end="")
                    tempIn = IO.realInput(n=dim)
                    if(tempIn != None):
                        break
                    points[i] = tempIn
                    i += 1

            elif (choice == 2):
                n, dim = IO.initPointsInput()
                points = p.generateRandomPoints(n, dim, -1e6, 1e6)

            elif (choice == 3):
                print("File format follows the input format")
                temp = IO.fileToPoints()
                if(temp == None):
                    print("file read error, returning to menu.\n")
                    continue
                n = temp[0]
                dim = temp[1]
                points = temp[2]

            # else:
            #     continue

            # p.sort(points)

            runMode = IO.getChoices(["Divide and Conquer Algorithm",
"Brute-force Algorithm", "Run both algorithm (benchmark)"], "run mode")

            if(runMode == 1 or runMode == 3):
                start = time.time()
                DNCresult = cp.findNearestPair(points)

```

```

end = time.time()

print('Divide and Conquer getDistance() calls count :',
p.counter.count)
print('Divide and Conquer execution time:', (end - start) *
(10 ** 3), 'ms')
print(f"Closest points distance = {DNCresult[0]}\n")

if(dim == 3):
    IO.plot3DPoints(points, DNCresult[1][0],
DNCresult[1][1])
elif(dim == 2):
    IO.plot2DPoints(points, DNCresult[1][0],
DNCresult[1][1])

    if(runMode == 2 or runMode == 3):
        start = time.time()
        BFresult = cp.findNearestPair(points, 1)
        end = time.time()
        print('Brute Force getDistance() calls count :',
p.counter.count)
        print('Brute Force execution time:', (end - start) * (10 **
3), 'ms')
        print(f"Closest points distance = {BFresult[0]}")

        if(dim == 3):
            IO.plot3DPoints(points, BFresult[1][0],
BFresult[1][1])
        elif(dim == 2):
            IO.plot2DPoints(points, BFresult[1][0],
BFresult[1][1])

        else:
            break
# End while
# -- Main program ends --

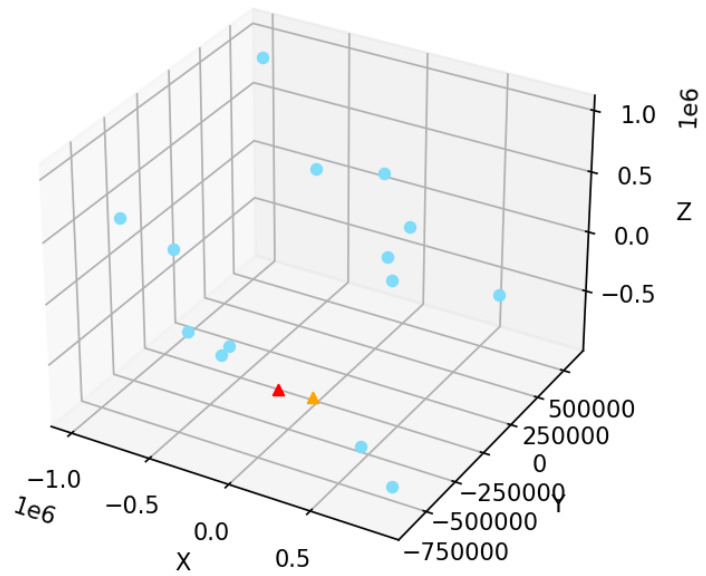
```

BAB IV

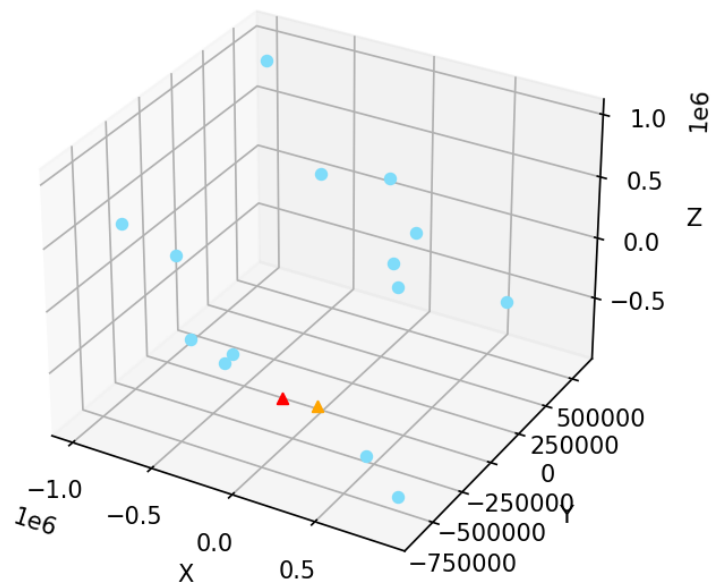
Eksperimen

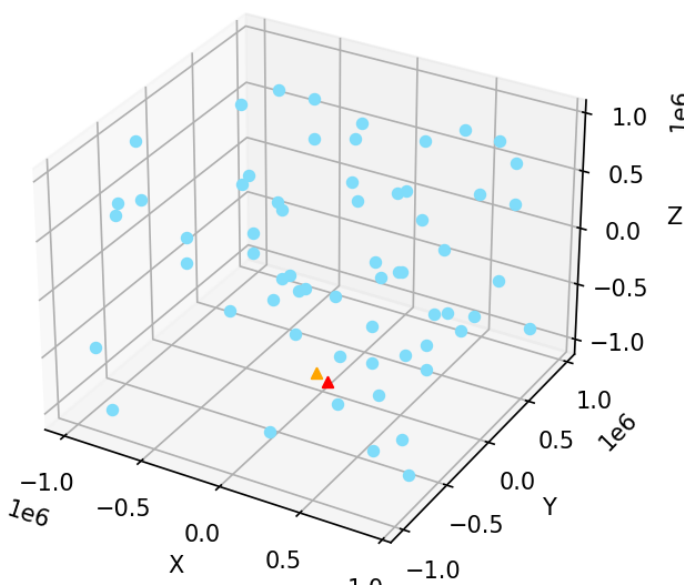
Berikut adalah hasil percobaan beberapa kasus uji terhadap program yang telah dibuat.

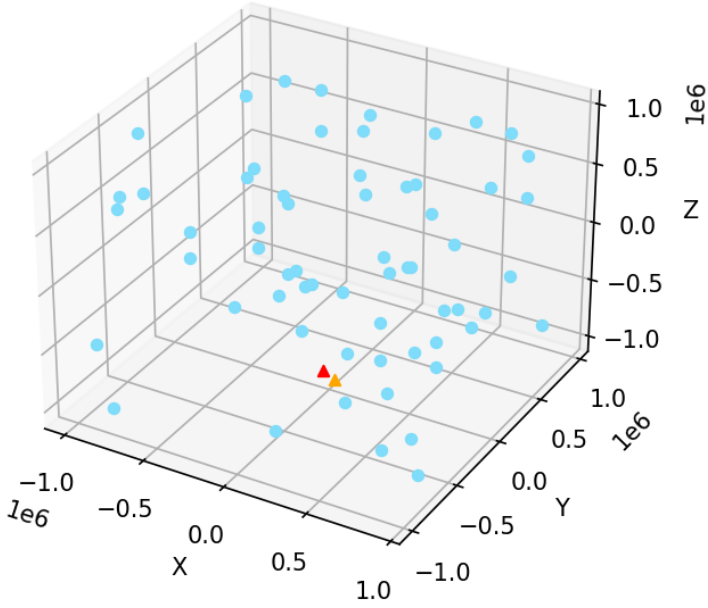
No .	Input/ Output	Gambar
1	Input	<pre> Nearest Points Welcome to nearest points calculator! Select action : [1] Start [2] Exit Your input : 1 Select input mode : [1] Console [2] Random [3] File [4] Return Your input : 2 ----- Input format: N {N is the number of points} D {D is the dimension of the points} ----- Input number of points: 16 Input dimensions: 3 Select run mode : [1] Divide and Conquer Algorithm [2] Brute-force Algorithm [3] Run both algorithm (benchmark) Your input : 3 </pre>
	Output	<pre> Your input : 3 Divide and Conquer getDistance() calls count : 19 Divide and Conquer execution time: 0.0 ms Closest points distance = 180549.75764356917 [Waiting the plot to be closed] </pre>

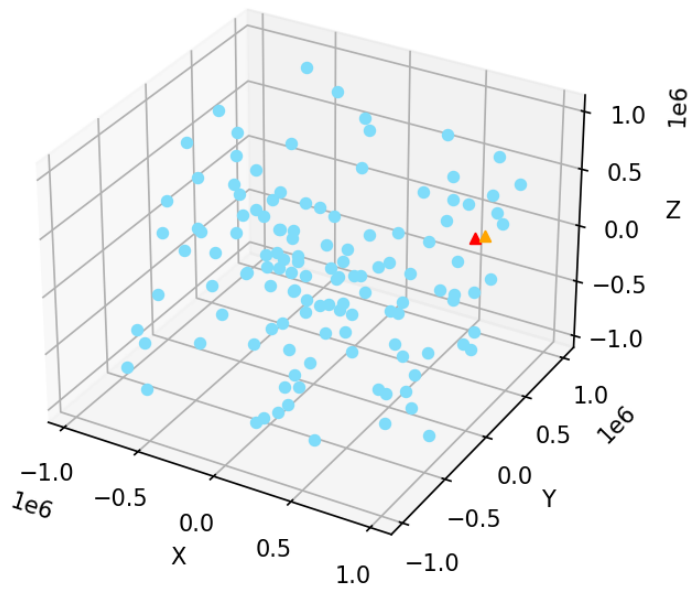


```
Brute Force getDistance() calls count : 120  
Brute Force execution time: 0.0 ms  
Closest points distance = 180549.75764356917  
[Waiting the plot to be closed]
```



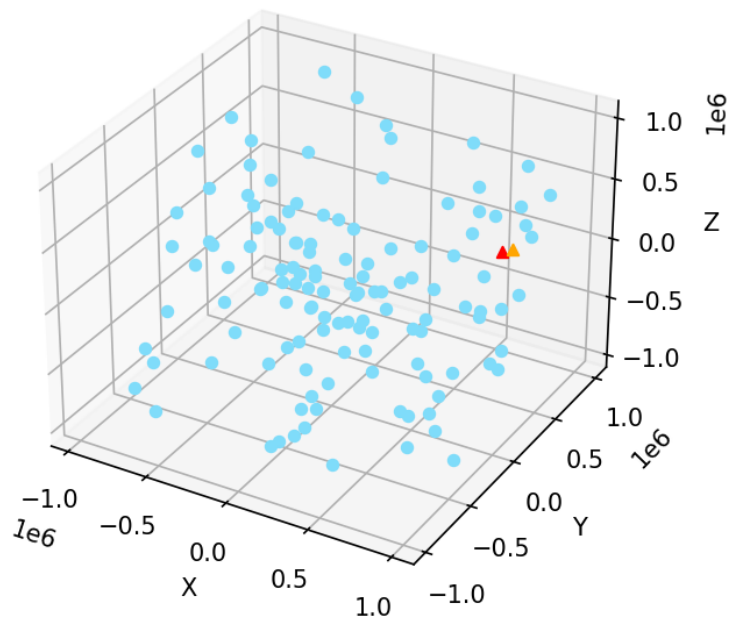
2	Input	<pre> Select action : [1] Start [2] Exit Your input : 1 Select input mode : [1] Console [2] Random [3] File [4] Return Your input : 2 ----- Input format: N {N is the number of points} D {D is the dimension of the points} ----- Input number of points: 64 Input dimensions: 3 Select run mode : [1] Divide and Conquer Algorithm [2] Brute-force Algorithm [3] Run both algorithm (benchmark) Your input : 3 </pre>
	Output	<pre> Divide and Conquer getDistance() calls count : 82 Divide and Conquer execution time: 1.0030269622802734 ms Closest points distance = 89237.91820959176 [Waiting the plot to be closed] </pre>  <pre> [Waiting the plot to be closed] Brute Force getDistance() calls count : 2016 Brute Force execution time: 3.002166748046875 ms Closest points distance = 89237.91820959176 [Waiting the plot to be closed] </pre>

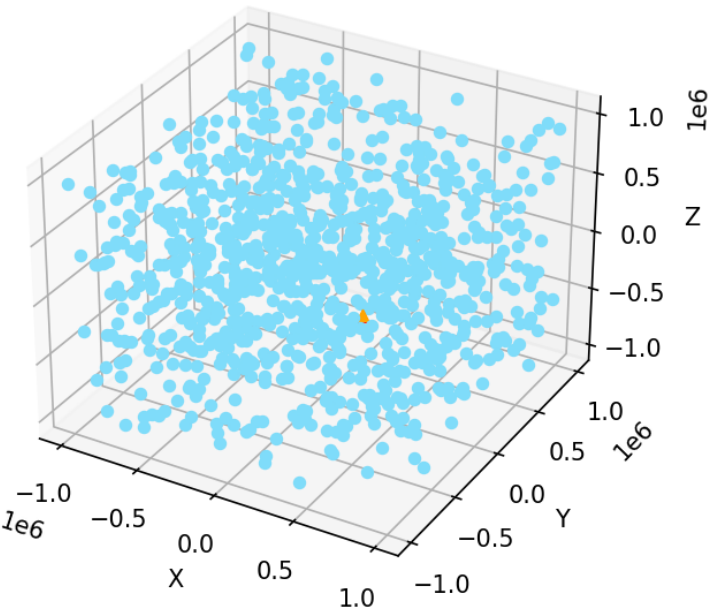
		
3	Input	<pre>Select action : [1] Start [2] Exit Your input : 1 Select input mode : [1] Console [2] Random [3] File [4] Return Your input : 2 ----- Input format: N {N is the number of points} D {D is the dimension of the points} ----- Input number of points: 128 Input dimensions: 3 Select run mode : [1] Divide and Conquer Algorithm [2] Brute-force Algorithm [3] Run both algorithm (benchmark) Your input : 3</pre>
	Output	<pre>Divide and Conquer getDistance() calls count : 168 Divide and Conquer execution time: 1.1382102966308594 ms Closest points distance = 76949.26544019693 [Waiting the plot to be closed]</pre>



```
Brute Force getDistance() calls count : 8128  
Brute Force execution time: 10.00070571899414 ms  
Closest points distance = 76949.26544019693
```

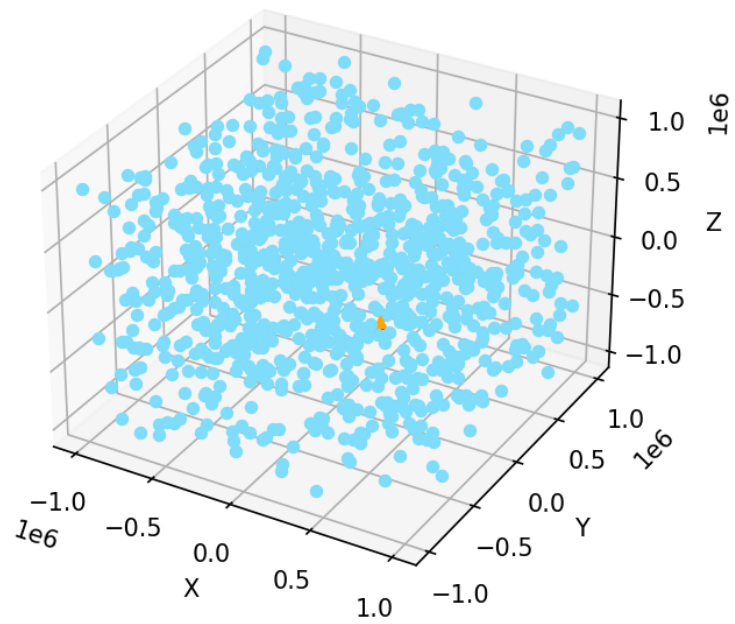
```
[Waiting the plot to be closed]
```



4	<i>Input</i>	<pre>Select action : [1] Start [2] Exit Your input : 1 Select input mode : [1] Console [2] Random [3] File [4] Return Your input : 2 ----- Input format: N {N is the number of points} D {D is the dimension of the points} ----- Input number of points: 1000 Input dimensions: 3 Select run mode : [1] Divide and Conquer Algorithm [2] Brute-force Algorithm [3] Run both algorithm (benchmark) Your input : 3</pre>
	<i>Output</i>	<pre>Divide and Conquer getDistance() calls count : 1368 Divide and Conquer execution time: 17.018556594848633 ms Closest points distance = 9362.11546036622 [Waiting the plot to be closed]</pre> 

```
[Waiting the plot to be closed]
Brute Force getDistance() calls count : 499500
Brute Force execution time: 517.0166492462158 ms
Closest points distance = 9362.11546036622
```

```
[Waiting the plot to be closed]
```



Lampiran

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

Link repository Github : https://github.com/Enliven26/Tucil2_13521138_13521148