LAPORAN TUGAS KECIL 2

IF2211/Strategi Algoritma 2022/2023

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

Muhammad Farrel Danendra Rachim / 13521048

Naufal Syifa Firdaus / 13521050

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2022/2023

Daftar Isi

1. Algoritma dan Masalah

A. Divide and Conquer

Algoritma *Divide and Conquer* adalah salah satu strategi algoritma untuk menyelesaikan persoalan dengan kompleksitas yang disederhanakan. Proses penyederhanaan masalah dibagi kedalam tiga tahap yang sesuai dengan namanya:

1. *Divide*
   Masalah yang akan diselesaikan dibagi menjadi beberapa persoalan yang lebih kecil yang mirip dengan masalah semula.
2. *Conquer*
   Masalah yang telah dibagi dan sekarang berukuran kecil, kini diselesaikan satu persatu.
3. *Combine*
   Solusi permasalah kecil digabung sehingga membentuk solusi persoalan semula.

Dengan langkah diatas kompleksitas penyelesaian masalah dapat direduksi drastis dibandingkan algoritma *Brute Force*. Walaupun kode yang dibuat cenderung lebih rumit dan sulit, efisiensi yang jauh meningkat dinilai sepadan dengan usaha yang dilakukan untuk menyusun kode programnya.

B. Persoalan

Persoalan yang akan diselesaikan adalah mencari dua titik dengan jarak terdekat di sebuah ruang 3 dimensi. Titik mempunyai koordinat x, y, dan z sebagai penanda posisinya di dalam ruang. 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 P1 = (x1, y1, z1) dan P2 = (x2, y2, z2) dihitung dengan rumus Euclidean berikut:

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

Selain itu, program dapat mencari sepasang titik terdekat untuk sekumpulan vektor di R^n, setiap vektor dinyatakan dalam bentuk x = (x1, x2,…, xn).

Penulis menggunakan algoritma *divide and conquer* untuk menyelesaikan permasalahan tersebut dan algoritma *brute force* untuk membandingkan permasalahan tersebut.

C. Implementasi Penyelesaian

Implementasi algoritma *Divide and Conquer* untuk menyelesaikan masalah yang diberikan dirumuskan dengan langkah-langkah sebagai berikut:

1. Setiap titik dalam sebuah ruang 3 dimensi direpresentasikan dengan sebuah array berurut berdasarkan koordinat x. Sehingga titik dengan koordinat x paling kecil berada di posisi array ke 0.
2. Bagi *array of point* tepat ditengah sehingga dihasilkan dua array (kiri dan kanan) dengan jumlah elemen sama atau berbeda hanya satu.
3. Secara rekursif bagi sisi kiri dan kanan dengan cara diatas. Basis dari rekursif tersebut adalah jika jumlah elemen pada array hanya 2 atau 3 sehingga mengembalikan pasangan dengan jarak terdekat (dapat menggunakan kondisional).
4. Dengan langkah sebelumnya didapat pasangan titik dengan jarak terdekat dari sisi kiri dan kanan, bandingkan keduanya untuk mendapat pasangan terdekat. Namun, pasangan tersebut belum tentu memiliki jarak paling dekat secara keseluruhan.
5. Buatlah sebuah bidang tipis imajiner dengan nilai koordinat sumbu x sama dengan nilai sumbu x titik pembagi pada langkah 2.
6. Buatlah *array of point* baru yang berisi semua titik pada ruang dengan jarak ke bidang tipis lebih kecil sama dengan jarak pasangan titik paling dekat yang diperoleh di langkah 4.
7. Hitung jarak semua titik di *array of point* yang baru satu sama lain sampai ditemukan jarak yang lebih dekat dari pasangan titik pada langkah 4 atau titik di array habis. Perhitungan pada langkah ini dapat menggunakan metode *Brute Force* karena jumlah titik yang dekat dengan bidang dijamin secara matematis menurun secara signifikan.

Kompleksitas algoritma dengan implementasi *Divide and Conquer* adalah $O(n\ log^2 n)$. Jika dibandingkan dengan algoritma *Brute Force* yang akan memiliki kompleksitas $O(n^2)$, algoritma diatas jauh lebih efisien dalam mencari pasangan titik terdekat.

2. *Source Code*

Program ini terbagi menjadi 4 file: dot.py, tool.py, dotVisualizer.py, dan main.py dengan deskripsi sebagai berikut:

A. File dot.py

Berisi kelas Dot yang merepresentasikan titik pada ruang 3D dan kelas PairOfDots yang merepresentasikan sebuah pasangan titik.

i. Daftar library
Kami memakai library __future__ untuk membuat parameter menjadi tipe kelas yang sama di beberapa metode di kelas Dot dan math untuk menghitung rumus Euclidean dua buah titik pada ruang 3D serta random untuk menghasilkan titik-titik dengan koordinat acak.

```
from __future__ import annotations
from math import sqrt
from random import *
```

ii. Kelas Dot
Kelas Dot memiliki atribut berupa x, y, z yang merepresentasikan koordinat suatu titik pada ruang 3D. Dot akan diinisialisasi dengan restriksi masing-masing sumbu antara -100000 sampai 100000 (nilai x, y, dan z dalam tiap titik berada di dalam *range* tersebut).

Metode	**Peran**
displayDot()	Meng-output dot dalam format (x, y, z)
distanceTwoDots(otherDot:Dot)	Menghitung jarak antara dua buah dot
isEqual(otherDot:Dot)	Menentukan apakah dua dot berlokasi di koordinat yang sama atau tidak

```python
class Dot:

    def __init__(self, dimention):
        #Inisialisasi dot
        self.coordinates = []

        for i in range(dimention):
            self.coordinates.append(round(uniform(-100000, 100000), 3))


    def displayDot(self):
        print("(", end="")
        for i in range (len(self.coordinates)-1):
            print("{x},".format(x = self.coordinates[i]),  end="")

        print("{y}".format(y=self.coordinates[i+1]), end="")
        print(")")

    def distanceTwoDots(self, otherDot: Dot):
        sumAll = 0

        for i in range(len(self.coordinates)):
            sumAll += (self.coordinates[i] - otherDot.coordinates[i])**2

        return sqrt(sumAll)

    def isEqual(self, otherDot: Dot):
        # Menentukan apakah dua dot berlokasi di koordinat yang sama atau tidak
        # (x1, y1, z1) = (x2, y2, z2)
        equal = True

        for i in range(len(self.coordinates)):
            if(self.coordinates[i] != otherDot.coordinates[i]):
                equal = False

        return equal
```

iii. Kelas PairOfDots

Kelas PairOfDots memiliki atribut berupa p1 dan p2 yang merepresentasikan dengan dua buah titik bertipe Dot, serta distance, yakni jarak antara kedua buah titik tersebut.

```python
class PairOfDots:
    def __init__(self, p1, p2, distance):
        self.p1 = p1
        self.p2 = p2
        self.distance = distance
```

B. File tool.py

Dalam file ini kami mengimplementasikan fungsi-fungsi yang digunakan untuk membuat dan menyelesaikan permasalahan yang diberi.

1. dotSorted

Keterangan	Parameter
Menghasilkan list berisi dot dengan nilai float (x,y,z) yang acak sebanyak n yang terurut berdasarkan nilai x.	n = jumlah titik yang akan dibentuk dimension = jumlah dimensi pada ruang

```python
def dotSorted(n, dimension):
    # Menghasilkan list point yang terurut berdasarkan nilai sumbu x

    pointList = []

    for i in range(n):
        dot = Dot(dimension)
        pointList.append(dot)

    pointList = sorted(pointList, key=lambda Dot:Dot.coordinates[0])

    return pointList
```

2. bruteForceClosest

Keterangan	Parameter
Mencari pasangan titik terdekat dengan menggunakan algoritma brute force.	pointList = *array of point* yang berisi semua titik dalam ruang

```python
def bruteforceClosest(pointList):

    euclideanCount = 0
    if(len(pointList) > 1):
        euclideanCount += 1
        closestPair = PairOfDots(pointList[0],
                                 pointList[1],
                                 pointList[0].distanceTwoDots(pointList[1]))

        for i in pointList:
                for j in pointList:
                    euclideanCount += 1
                    if (i.distanceTwoDots(j) < closestPair.distance and i != j):
                        closestPair.p1 = i
                        closestPair.p2 = j
                        closestPair.distance = i.distanceTwoDots(j)

    return closestPair, euclideanCount
```

3. minDistancePair

Keterangan	Parameter
Menentukan pasangan titik dengan jarak paling dekat antara dua pasangan titik.	Pair1 = pasangan titik pertama Pair2 = pasangan titik kedua

```
def minDistancePair(pair1, pair2):

    if(pair1.distance < pair2.distance):
        return pair1
    else:
        return pair2
```

4. distanceAbsis

Keterangan	Parameter
Menghitung jarak antara dua titik dengan hanya menggunakan koordinat x dari kedua titik. Digunakan untuk mencari jarak titik kepada suatu bidang.	point1 = titik pertama point2 = titik referensi sebuah bidang

```
def distanceAbsis(point1, point2):
    # Menghitung jarak dengan memperhatikan atribut x saja
    return abs(point1.coordinates[0] - point2.coordinates[0])
```

5. calculateClosest

Keterangan	Parameter
Menghasilkan pasangan titik yang paling dekat menggunakan algoritma Divide and Conquer.	pointList = *array of list* yang berisi semua titik dalam ruang yang terurut berdasarkan koordinat x.

```python
def calculateClosest(pointList):  # bidang pembatas ditengah (x, 0, 0)

    # BASIS
    countEuclidean = 0
    if(len(pointList) == 2):

        closestPair = PairOfDots(pointList[0],
                                 pointList[1],
                                 pointList[0].distanceTwoDots(pointList[1]))

        countEuclidean += 1
        return closestPair, countEuclidean

    elif (len(pointList) == 3):

        closestPair = PairOfDots(pointList[0],
                                 pointList[1],
                                 pointList[0].distanceTwoDots(pointList[1]))

        if(pointList[0].distanceTwoDots(pointList[2]) < closestPair.distance):
            closestPair.p2 = pointList[2]
            closestPair.distance = pointList[0].distanceTwoDots(pointList[2])

        elif (pointList[1].distanceTwoDots(pointList[2]) < closestPair.distance):
            closestPair.p1 = pointList[1]
            closestPair.p2 = pointList[2]
            closestPair.distance = pointList[1].distanceTwoDots(pointList[2])

        countEuclidean += 3

        return closestPair, countEuclidean

    # REKURENS
    else:
        middle = len(pointList) // 2

        closestPair_Right, countRightEuclidean = calculateClosest(pointList[:middle])
        countEuclidean += countRightEuclidean
        closestPair_Left, countLeftEuclidean = calculateClosest(pointList[middle:])
        countEuclidean += countLeftEuclidean

        closestPair = minDistancePair(closestPair_Left, closestPair_Right)

        # EVALUASI BIDANG
        # bidang yang mengandung poin dengan kemungkinan
        # jarak lebih kecil dari closestPair
        pointField = []

        for i in pointList:
            if (distanceAbsis(i, pointList[middle]) < closestPair.distance):
                pointField.append(i)

        for a in pointField:
            for b in pointField:
                countEuclidean += 1
                if (a.distanceTwoDots(b) < closestPair.distance and a != b):
                    closestPair = PairOfDots(a, b, a.distanceTwoDots(b))

        return closestPair, countEuclidean
```

6. getSeparateXYZ

Keterangan	**Parameter**
Mengembalikan tuple berisi list atribut x, y, z yang terpisah dari titik-titik yang bukan merupakan pasangan titik terdekat.	pointList = *array of list* yang berisi semua titik dalam ruang closestPair = pasangan titik paling dekat

```
def getSeparateXYZ(pointList, closestPair):
    # Mengembalikan tuple berisi list atribut x, y, z yang terpisah
    # dari titik-titik yang bukan merupakan pasangan titik terdekat
    xs = []
    ys = []
    zs = []
    for i in range (0, len(pointList)):
        if (not pointList[i].isEqual(closestPair.p1)):
            if (not pointList[i].isEqual(closestPair.p2)):
                xs.append(pointList[i].x)
                ys.append(pointList[i].y)
                zs.append(pointList[i].z)
    return xs, ys, zs
```

7. getSeperateClosestXYZ

Keterangan	**Parameter**
Mengembalikan tuple berisi list atribut x, y, z yang terpisah dari titik-titik yang merupakan pasangan titik terdekat.	closestPair = pasangan titik paling dekat

```
def getSeparateClosestXYZ(closestPair):
    # Mengembalikan tuple berisi list atribut x, y, z yang terpisah
    # dari titik-titik yang merupakan pasangan titik terdekat
    xclosest = []
    yclosest = []
    zclosest = []
    xclosest.append(closestPair.p1.x)
    yclosest.append(closestPair.p1.y)
    zclosest.append(closestPair.p1.z)
    xclosest.append(closestPair.p2.x)
    yclosest.append(closestPair.p2.y)
    zclosest.append(closestPair.p2.z)
    return xclosest, yclosest, zclosest
```

C. File dotVisualizer.py

File program berikut berisi fungsi untuk memvisualisasikan ruang 3d beserta titik-titik di dalamnya.

1. Plot3d

Keterangan	Parameter
Menggambarkan semua titik dalam bidang 3D. Sepasang titik yang jaraknya terdekat ditunjukkan dengan warna merah, sedangkan titik lainnya ditunjukkan dengan warna hijau	x, y, z Closest = semua koordinat dari pasangan yang paling dekat. x,y,z others = semua koordinat dari titik dalam ruang.

```
def plot3D(xClosest, yClosest, zClosest, xOthers, yOthers, zOthers):
    # Menggambarkan semua titik dalam bidang 3D
    # Sepasang titik yang jaraknya terdekat ditunjukkan dengan warna merah,
    # sedangkan titik lainnya ditunjukkan dengan warna hijau
    plt.rcParams["figure.figsize"] = [12, 9]
    plt.rcParams["figure.autolayout"] = True
    fig = plt.figure()
    axis = fig.add_subplot(projection="3d")
    axis.scatter(xOthers, yOthers,zOthers, c='green')
    axis.scatter(xClosest,yClosest,zClosest, c='red')
    plt.show()
```

D. File main.py

Berikut merupakan file yang digunakan untuk menjalankan keseluruhan program. Program didalamnya berisi interface dan penanganan input/output dari program.

1. Proses input
   Input pengguna berupa nilai n dan dimensi akan divalidasi sampai memiliki tipe data integer dengan n >= 2.

```
print("\n\nWelcome to Closest 3D Dot Finder!")
print("\n=============================================\n")

while True:
    n = input("Masukkan nilai n: ")
    try:
        n = int(n)
    except ValueError:
        print("Nilai n harus berupa integer >= 2. Silakan coba lagi.")
    else:
        try:
            if n < 2:
                raise InvalidN
            else:
                break
        except InvalidN:
            print("Nilai n >= 2. Silakan coba lagi.")

while True:
    dimension = input("Masukkan nilai dimension: ")
    try:
        dimension = int(dimension)
    except ValueError:
        print("Nilai dimension harus berupa integer >= 2. Silakan coba lagi.")
    else:
        try:
            if dimension < 2:
                raise InvalidN
            else:
                break
        except InvalidN:
            print("Nilai dimension >= 2. Silakan coba lagi.")

print("\n=============================================\n")
```

2. Algoritma
   Sebanyak n buah dot (titik) akan di-*generate* secara acak ke dalam sebuah list. Dari list tersebut akan dicari sepasang titik yang jaraknya paling dekat serta jumlah operasi Euclidean yang dilakukan dengan algoritma *divide and conquer* dan algoritma *brute force*. Masing-masing penyelesaian tersebut diukur waktunya menggunakan library timeit.

```
points = dotSorted(n, dimension)

startDVC = timer()
closestPairDVC, countDVC = calculateClosest(points)
endDVC = timer()
timeDVC = endDVC - startDVC

startBF = timer()
closestPairBrute, countBrute = bruteforceClosest(points)
endBF = timer()
timeBF = endBF - startBF
```

3. Output
   Akan ditampilkan pasangan paling dekat, jarak pasangan tersebut, banyak operasi Euclidean yang dilakukan, dan waktu penyelesaian dalam microsecond untuk masing-masing algoritma *divide and conquer* dan algoritma *brute force*.

```
print("\n=============================================\n")
print("***DIVIDE AND CONQUER METHOD***")
print("\n=============================================\n")

print("Closest Pair: ")
closestPairDVC.p1.displayDot()
closestPairDVC.p2.displayDot()

print("\n=============================================\n")

print("Pair Distance:")
print(closestPairDVC.distance)

print("\n=============================================\n")

print("The amount of Euclidean operations done:")
print(countDVC)

print("\n=============================================\n")
print("Divide and Conquer algorithm duration: ")
print(f"{timeDVC * 10**6 :.3f} microseconds ({timeDVC :.f} seconds)")
print("\n=============================================\n")
```

```
print("\n=============================================\n")
print("***BRUTE FORCE METHOD***")
print("\n=============================================\n")

print("Closest Pair: ")
closestPairBrute.p1.displayDot()
closestPairBrute.p2.displayDot()

print("\n=============================================\n")

print("Pair Distance:")
print(closestPairBrute.distance)

print("\n=============================================\n")

print("The amount of Euclidean operations done:")
print(countBrute)

print("\n=============================================\n")
print("Brute Force algorithm duration: ")
print(f"{timeBF * 10**6 :.3f} microseconds ({timeBF :.2f} seconds)")
print("\n=============================================\n")
```

Diperoleh tiga buah list, masing-masing berisi x, y, dan z, untuk semua titik yang bukan termasuk pasangan titik terdekat, serta tiga list lain berisi x, y, dan z untuk pasangan titik terdekat. Akhirnya, akan ditampilkan grafik 3D yang menampilkan semua titik.

```
xOthers, yOthers, zOthers = getSeparateXYZ(points,closestPairDVC)
xClosest,yClosest, zClosest = getSeparateClosestXYZ(closestPairDVC)

plot3D(xClosest,yClosest,zClosest,xOthers,yOthers,zOthers)
```

3. Eksperimen

Eksperimen dilakukan untuk mengetahui hasil keluaran implementasi algoritma *Divide and Conquer* dan kebenarannya serta keefektifannya yang diukur menggunakan durasi eksekusi algoritma dalam program.

Keseluruhan eksperimen dilakukan di komputer dengan spesifikasi:

1. Laptop : HP Pavilion Gaming 15
2. Processor : Intel(R) Core(TM) i5-10300H CPU @ 2.50 GHz
3. RAM : 16 GB @ 2933 MHz
4. Storage : SSD 512 Gb

a. Eksperimen dengan n = 16 dan dimension = 3

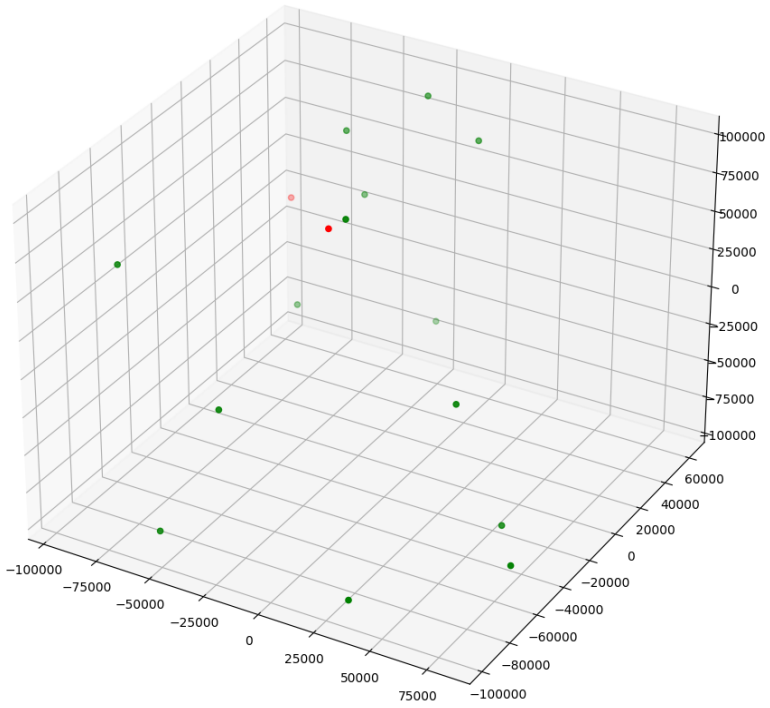
iv. Input

```
Welcome to Closest 3D Dot Finder!

==========================================

Masukkan nilai n: 16
Masukkan nilai dimension: 3

==========================================
```

v. Output

```
==========================================

***DIVIDE AND CONQUER METHOD***

==========================================

Closest Pair:
(-67381.166,16991.361,37981.4)
(-45430.111,10407.447,30829.132)

==========================================

Pair Distance:
24007.325480282994

==========================================

The amount of Euclidean operations done:
155

==========================================

Divide and Conquer algorithm duration:
231.300 microseconds (0.00 seconds)

==========================================
```

```
==========================================

***BRUTE FORCE METHOD***

==========================================

Closest Pair:
(-67381.166,16991.361,37981.4)
(-45430.111,10407.447,30829.132)

==========================================

Pair Distance:
24007.325480282994

==========================================

The amount of Euclidean operations done:
257

==========================================

Brute Force algorithm duration:
358.500 microseconds (0.00 seconds)

==========================================
```



b. Eksperimen dengan n = 64 dan dimension = 3

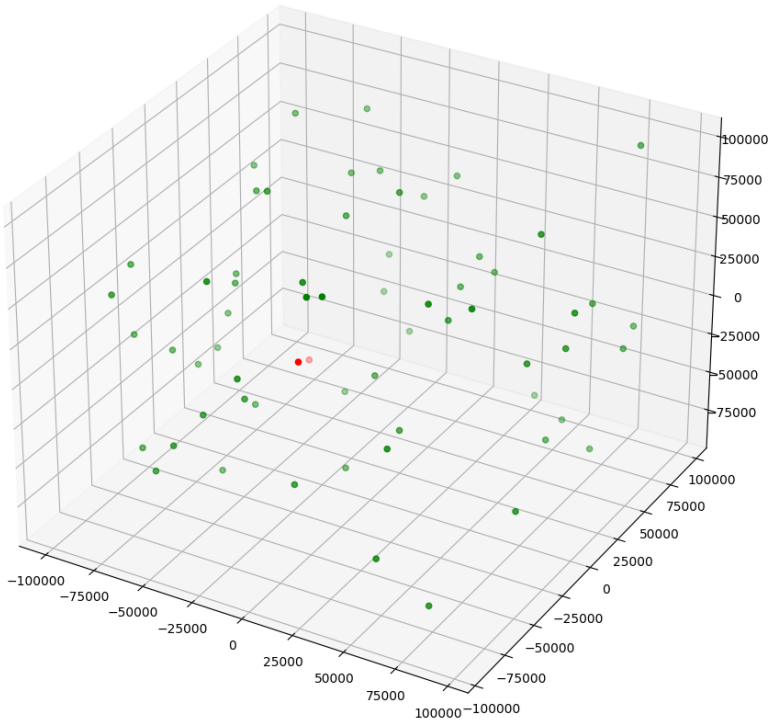
i. Input

```
Welcome to Closest 3D Dot Finder!

==========================================

Masukkan nilai n: 64
Masukkan nilai dimension: 3

==========================================
```

ii. Output

```
==========================================

***DIVIDE AND CONQUER METHOD***

==========================================

Closest Pair:
(-23886.108,-20750.085,-5818.828)
(-20588.083,-17183.614,-5563.71)

==========================================

Pair Distance:
4864.336490045686

==========================================

The amount of Euclidean operations done:
1139

==========================================

Divide and Conquer algorithm duration:
2316.300 microseconds (0.00 seconds)

==========================================
```

```
==========================================

***BRUTE FORCE METHOD***

==========================================

Closest Pair:
(-23886.108,-20750.085,-5818.828)
(-20588.083,-17183.614,-5563.71)

==========================================

Pair Distance:
4864.336490045686

==========================================

The amount of Euclidean operations done:
4097

==========================================

Brute Force algorithm duration:
4867.400 microseconds (0.00 seconds)

==========================================
```



c. Eksperimen dengan n = 128 dan dimension = 3

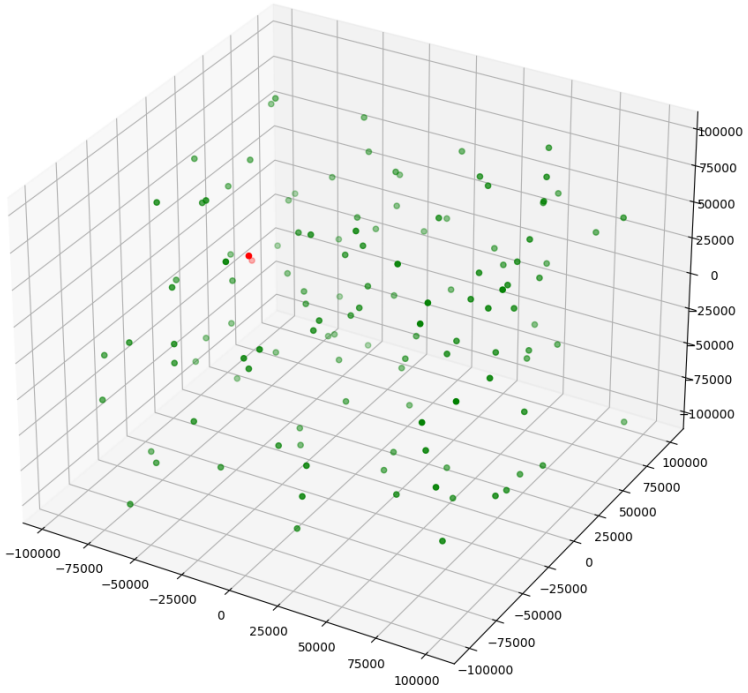
i. Input

```
Welcome to Closest 3D Dot Finder!

==========================================

Masukkan nilai n: 128
Masukkan nilai dimension: 3

==========================================
```

ii. Output

```
==========================================

***DIVIDE AND CONQUER METHOD***

==========================================

Closest Pair:
(-67119.784,17387.754,6344.112)
(-65496.245,17452.196,3629.797)

==========================================

Pair Distance:
3163.469230940929

==========================================

The amount of Euclidean operations done:
3330

==========================================

Divide and Conquer algorithm duration:
3917.700 microseconds (0.00 seconds)

==========================================
```

```
==========================================

***BRUTE FORCE METHOD***

==========================================

Closest Pair:
(-67119.784,17387.754,6344.112)
(-65496.245,17452.196,3629.797)

==========================================

Pair Distance:
3163.469230940929

==========================================

The amount of Euclidean operations done:
16385

==========================================

Brute Force algorithm duration:
17567.300 microseconds (0.02 seconds)

==========================================
```



d. Eksperimen dengan n = 1000 dan dimension = 3

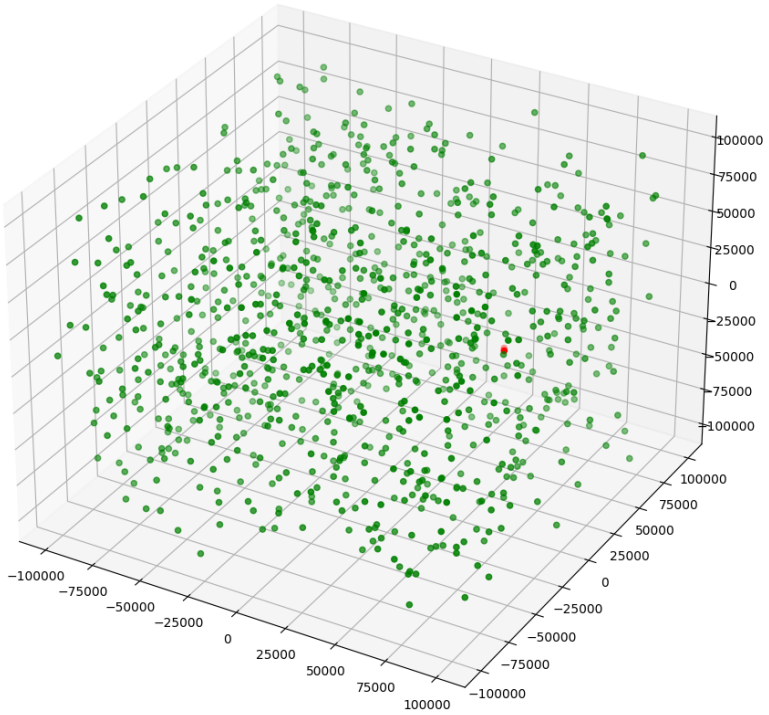
i. Input

```
Welcome to Closest 3D Dot Finder!

================================================

Masukkan nilai n: 1000
Masukkan nilai dimension: 3

================================================
```

ii. Output

```
==========================================

***DIVIDE AND CONQUER METHOD***

==========================================

Closest Pair:
(18725.809,96325.97,-70515.918)
(19365.456,95262.621,-71402.79)

==========================================

Pair Distance:
1525.2545114812742

==========================================

The amount of Euclidean operations done:
103079

==========================================

Divide and Conquer algorithm duration:
104443.600 microseconds (0.10 seconds)

==========================================
```

```
==========================================

***BRUTE FORCE METHOD***

==========================================

Closest Pair:
(18725.809,96325.97,-70515.918)
(19365.456,95262.621,-71402.79)

==========================================

Pair Distance:
1525.2545114812742

==========================================

The amount of Euclidean operations done:
1000001

==========================================

Brute Force algorithm duration:
974430.000 microseconds (0.97 seconds)

==========================================
```



4. Kesimpulan dan Saran

A. Kesimpulan

Dari tugas kecil 2 ini, kami memperoleh kesimpulan bahwa kami dapat menemukan pasangan titik terdekat dalam ruang 3D dengan menggunakan algoritma *brute force* dan *divide and conquer*. Namun, pencarian dengan algoritma *divide and conquer* lebih efisien karena memiliki kompleksitas waktu sebesar $O(n\,log^2 n)$, dibandingkan algoritma *brute force* yang memiliki kompleksitas waktu $O(n^2)$. Perbedaan kompleksitas waktu ini akan menjadi lebih jelas dan terbukti dari eksperimen kami jika jumlah titik mencapai ratusan bahkan ribuan.

B. Saran

Algoritma pencarian pasangan titik terdekat dapat dikembangkan lagi sehingga waktu pencarian dapat menjadi lebih singkat. Kami juga masih dapat membuat performa program ini menjadi lebih efisien dengan memakai bahasa pemrograman lain.

5. Lampiran

Link repository: https://github.com/Breezy-DR/Tucil2_13521048_13521050

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