LAPORAN TUGAS KECIL 2
IF-2211 STRATEGI ALGORITMA

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

Disusun oleh:

Christian Albert Hasiholan	**13521078**
Reza Pahlevi Ubaidillah	**13521165**

PROGRAM STUDI TEKNIK INFORMATIKA

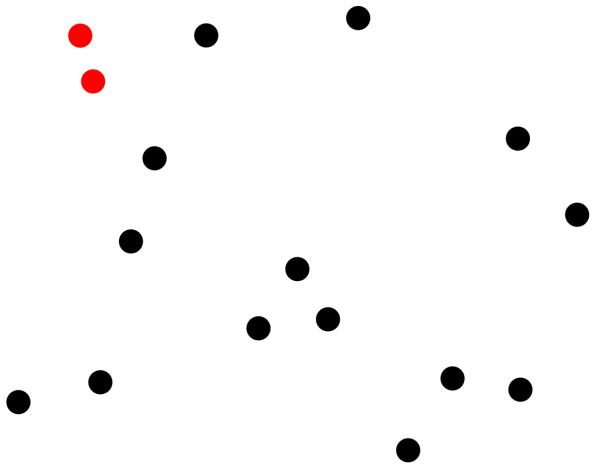
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

BAB I
DESKRIPSI MASALAH

Diberikan himpunan titik, P, yang terdiri dari n buah titik pada bidang 2-D. Maka perlu ditentukan sepasang titik dalam P yang jaraknya terdekat satu sama lain.



Pada Tucil 2 ini, algoritma untuk memecahkan persoalan di atas dapat dikembangkan untuk mencari sepasang titik terdekat pada bidang 3D. Misalkan terdapat n buah titik pada ruang 3D, dimana setiap titik P di dalam ruang tersebut dinyatakan dengan koordinat $P = (x, y, z)$. Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain . Jarak dari tiap titik, $P_1 = (x_1, y_1, z_1)$ dan $P_2 = (x_2, y_2, z_2)$, dapat dihitung dengan rumus Euclidean berikut:

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

Buatlah program untuk mencari sepasang titik dengan jarak terdekat satu sama lain dengan menerapkan algoritma *Divide and Conquer* untuk penyelesaiannya, dan bandingkanlah dengan algoritma *Brute Force*.

BAB II

ALGORITMA

1. Input

   Pada awal berjalan, program akan meminta input berupa `nPoint` banyak titik yang akan di-generate dan `nDim` dimensi yang digunakan. Kemudian program akan memanggil fungsi `initializePoint(nPoint, nDim)` untuk menghasilkan senarai titik berdimensi `nDim` sebanyak `nPoint` secara acak dengan posisi titik dalam rentang -100 hingga 100 pada tiap sumbunya menggunakan fungsi `uniform(-100, 100)` dari modul random *built-in* python.

```python
#inisialisasi point secara random
def initializePoint(nPoint, nDimension):
    listOfPoints = []
    i = 0
    for i in range(nPoint):
        temp = []
        j = 0
        for j in range(nDimension):
            value = random.uniform(-100, 100)
            temp.append(value)

        listOfPoints.append(temp)

    return listOfPoints
```

2. Solusi Brute Force

   Solusi jarak terpendek dengan Brute Force diperoleh dengan menghitung jarak setiap pasang titik yang mungkin, berarti program akan melakukan perhitungan jarak euclidean

sebanyak $C_2^n = \frac{n(n-1)}{2}$ kali. Dengan demikian, kompleksitas algoritma solusi ini adalah $O(n^2)$.

```
#menentukan pasangan terdekat dengan Brute Force
def closestPairBF(listOfPoints, filter = None):
    nPoints = len(listOfPoints)
    nDim = len(listOfPoints[0])
    counter = 0

    d = eucDistance(listOfPoints[0], listOfPoints[1], nDim)
    pair1 = []
    pair2 = []
    i = 0
    for i in range(nPoints):
        for j in range(i+1,nPoints):
            if(filter!=None and filter(listOfPoints[i], listOfPoints[j])):
                continue
            distResult = eucDistance(listOfPoints[i], listOfPoints[j], nDim)
            counter += 1
            if d > distResult:
                d = distResult
                pair1 = listOfPoints[i]
                pair2 = listOfPoints[j]

    return d, pair1, pair2, counter
```

3. Merge sort

Algoritma sorting merge sort akan membagi senarai menjadi dua bagian sama besar (atau berbeda satu jika ganjil) (DIVIDE). Kemudian secara rekursif lakukan hal yang sama pada masing-masing bagian (CONQUER). Apabila senarai tunggal, berarti senarai sudah terurut (SOLVE). Kemudian gabungkan kedua bagian sehingga diperoleh senarai sebelum pembagian yang terurut (MERGE). Pemrosesan merge memerlukan operasi sebanyak $cn = O(n)$. Dengan demikian, kompleksitas algoritmanya dapat dinyatakan sebagai $T(n) = a$ untuk n=1 dan $T(n) = 2T(n/2) + cn$ untuk $n > 1$. Sesuai relasi rekurens

Teorema Master, $T(n) = aT(n/b) + cn^d$ di mana $a = 2, b = 2, d = 1$, kompleksitas algoritmanya adalah $O(nlogn)$.

```python
#untuk menggabungkan 2 list secara terurut
def merge(list1, list2, axis): #axis = pengurutan point berdasarkan sumbu
    result = []
    n1 = len(list1)
    n2 = len(list2)

    i = 0
    j = 0
    while ((i < n1) and (j < n2)):
        if list1[i][axis] <= list2[j][axis]:
            result.append(list1[i])
            i += 1
        else:
            result.append(list2[j])
            j += 1

    while (i < n1):
        result.append(list1[i])
        i += 1

    while (j < n2):
        result.append(list2[j])
        j += 1

    return result


#sort dengan Divide and Conquer
def mergeSort(listOfPoints, axis):
    n = len(listOfPoints)
    mid = n//2

    if (n > 1):
        list1 = listOfPoints[0:mid]
```

```
        list2 = listOfPoints[mid:]

        sorted1 = mergeSort(list1, axis)
        sorted2 = mergeSort(list2, axis)

        result = merge(sorted1, sorted2, axis)
    else:
        result = listOfPoints

    return result
```

4. Solusi Divide and Conquer

Solusi ini bekerja dengan cara membagi himpunan titik menjadi dua bagian (*S1* dan *S2*) dengan jumlah titik sama banyak (atau berbeda satu jika ganjil), berarti perlu dilakukan *sorting* berdasarkan satu sumbu dahulu supaya hal itu dapat tercapai (DIVIDE). Lakukan pembagian terus menerus secara rekursif pada masing-masing bagian untuk mencari jarak dan pasangan titik terdekat (CONQUER). Saat suatu himpunan titik memiliki paling banyak 3 titik, program akan menghitung jarak euclidean terpendek antara titik-titik dengan algoritma Brute Force, diperolehlah pasangan titik terdekat beserta jaraknya pada satu bagian (SOLVE). Ada 3 kemungkinan letak pasangan titik dengan jarak terpendek, yaitu kedua titik berada di *S1*, kedua titik berada di *S2*, atau satu titik berada di *S1* dan titik lain di *S2* (COMBINE).

Program dapat mencari jarak terpendek *d* antara *S1* dan *S2* dengan mudah, cukup dengan membandingkan nilainya. Lantas bagaimana cara membandingkan jarak antara dua titik yang berada pada bagian berbeda? Bisa saja langsung di-Brute Force, tapi kita tidak mau itu, yang jelas misalkan dua titik ini ada, jaraknya pastilah lebih kecil dari *d*. Oleh karena itu, program akan mencari pasangan titik terdekat dalam *strip*, yaitu himpunan titik pada *S1* dan *S2* yang terletak kurang dari *d* dari perbatasan. Setelah itu barulah pasangan titik terdekat bisa diperoleh dari hasil algoritma Brute Force pada titik-titik dalam *strip*, tetapi dengan syarat tidak ada satupun sumbu yang sama di antara dua titik memiliki jarak lebih dari *d*. Pemrosesan titik dalam *strip* memerlukan $cn = O(n)$. Kompleksitas algoritma ini dapat dinyatakan sebagai $T(n) = 2T(n/2) + cn$ untuk $n > 2$ dan $T(n) = a$ untuk $n = 2$.

Sesuai relasi rekurens Teorema Master, $T(n) = aT(n/b) + cn^d$ di mana $a = 2, b = 2, d = 1$, kompleksitas algoritmanya adalah $O(nlogn)$, lebih baik dari solusi Brute Force $O(n^2)$.

```
def closestPairDNC(listOfPoints, axis: int = Axis.X.value):
    n = len(listOfPoints)
    sortedPoints = mergeSort(listOfPoints, axis)

    if n <= 3:
        d, point1, point2, counter = closestPairBF(sortedPoints)
    else:
        halfn = n//2

        if (n%2 == 0):
            space1 = sortedPoints[0:halfn]
            space2 = sortedPoints[halfn:n]
            middleLine = (sortedPoints[halfn-1][axis] +
sortedPoints[halfn][axis]) / 2
        else:
            space1 = sortedPoints[0:halfn]
            space2 = sortedPoints[halfn+1:n]
            middleLine = sortedPoints[halfn][axis]

        leftDist, leftPoint1, leftPoint2, counter = closestPairDNC(space1)
        rightdist, rightPoint1, rightPoint2, temp =  closestPairDNC(space2)
        counter += temp

        if (leftDist < rightdist):
            d = leftDist
            point1 = leftPoint1
            point2 = leftPoint2
        else:
            d = rightdist
            point1 = rightPoint1
            point2 = rightPoint2
```

```
        stripPoint = stripList(sortedPoints, middleLine, d)
        sortedStrip = mergeSort(stripPoint, axis+1)

        stripDist, stripPoint1, stripPoint2, temp =
closestPairBF(sortedStrip, filter=lambda point1, point2: ignore(point1,
point2, d))
        counter += temp

        if (stripDist < d):
            d = stripDist
            point1 = stripPoint1
            point2 = stripPoint2


    return d, point1, point2, counter
```

BAB III

SOURCE CODE

Source code dari file **main.py**

```
import point
import time
import plot
import platform

if __name__ =="__main__":
    print("Selamat datang di program Find Closest Point!!!")

    while (True):
        print("\nMenu")
        print("1. 3 Dimensional Space")
        print("2. Euclidean Space (Vector Rn)")
        print("3. Keluar Program")

        try:
            menuInp = int(input("Masukan pilihan menu: "))

            if (menuInp not in [1,2,3]):
                print("Pilihan menu tidak ada. Pilih kembali menu!")
            else:
                if (menuInp == 3):
                    print("Terima kasih sudah menggunakan program kami. Sampai
jumpa")
                    break
                    exit()
                else:
                    if (menuInp == 1):
                        nDim = 3
                    else:
                        nDim = int(input("Dimensi: "))

                    nPoint = int(input("Banyak point: "))
```

```
                p = point.initializePoint(nPoint, nDim)

                print("Closest pair dengan Brute Force:")
                t1 = time.time()
                distBF, bf1, bf2, c1 = point.closestPairBF(p)
                t2 = time.time()
                print("Jarak terdekat", distBF)
                print("Titik 1:", bf1)
                print("Titik 2:", bf2)
                print("Waktu eksekusi (s):", t2-t1)
                print("Banyak operasi Euclidean Distance:", c1)
                print("Running on", platform.platform(), platform.processor())

                print("\nClosest pair dengan Divide and Conquer:")
                t1 = time.time()
                distDC, dc1, dc2, c2 = point.closestPairDNC(p)
                t2 = time.time()
                print("Jarak terdekat", distDC)
                print("Titik 1:", dc1)
                print("Titik 2:", dc2)
                print("Waktu eksekusi (s):", t2-t1)
                print("Banyak operasi Euclidean Distance:", c2)
                print("Running on", platform.platform(), platform.processor())

                if nDim==3:
                    plot.plot_3d(p, bf1, bf2)

        except:
            print("Input salah. Kembali ke menu!")
```

Source code dari **plot.py**

```
import matplotlib.pyplot as plt
from typing import List
from axis import Axis

def plot_3d(listOfPoints: List[List[float]], pair1: List[float], pair2:
List[float]) -> None:
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')

    for point in listOfPoints:
        if (all(a1==a2 for a1, a2 in zip(point, pair1)) or all(a1==a2 for a1,
a2 in zip(point, pair2))):
            color = '#EE4B2B'
        else:
            color = '#add8e6'
        ax.scatter(point[Axis.X.value], point[Axis.Y.value],
point[Axis.Z.value], marker='o', c=color)

    plt.show()
```

Source code dari **axis.py**

```
import enum

class Axis(enum.Enum):
    X = 0
    Y = 1
    Z = 2
```

Source code dari **point.py**

```python
import math
import random
from axis import Axis
from typing import List, Tuple

#hitung jarak 2 point
def eucDistance(point1, point2, dimension):
    sum = 0
    i = 0
    for i in range(dimension):
        sum += (point1[i] - point2[i])**2

    return math.sqrt(sum)


#inisialisasi point secara random
def initializePoint(nPoint, nDimension):
    listOfPoints = []
    i = 0
    for i in range(nPoint):
        temp = []
        j = 0
        for j in range(nDimension):
            value = random.uniform(-100, 100)
            temp.append(value)

        listOfPoints.append(temp)

    return listOfPoints


#untuk menggabungkan 2 list secara terurut
def merge(list1, list2, axis): #axis = pengurutan point berdasarkan sumbu
    result = []
```

```
    n1 = len(list1)
    n2 = len(list2)

    i = 0
    j = 0
    while ((i < n1) and (j < n2)):
        if list1[i][axis] <= list2[j][axis]:
            result.append(list1[i])
            i += 1
        else:
            result.append(list2[j])
            j += 1

    while (i < n1):
        result.append(list1[i])
        i += 1

    while (j < n2):
        result.append(list2[j])
        j += 1

    return result


#sort dengan Divide and Conquer
def mergeSort(listOfPoints, axis):
    n = len(listOfPoints)
    mid = n//2

    if (n > 1):
        list1 = listOfPoints[0:mid]
        list2 = listOfPoints[mid:]

        sorted1 = mergeSort(list1, axis)
        sorted2 = mergeSort(list2, axis)

        result = merge(sorted1, sorted2, axis)
    else:
```

```
        result = listOfPoints

    return result


#mendapat point-point yang berada di strip area
def stripList(listOfPoints, middle, d):
    result = []
    for point in listOfPoints:
        if abs(point[Axis.X.value] - middle) < d:
            result.append(point)

    return result


#mengabaikan titik yang jarak pada salah satu sumbu lebih dari d
def ignore(point1: List[float], point2: List[float], d: float) -> bool:
    for i in range(len(point1)):
        if abs(point1[i]-point2[i]) > d:
            return True
    return False


#menentukan pasangan terdekat dengan Divide and Conquer
def closestPairDNC(listOfPoints, axis: int = Axis.X.value):
    n = len(listOfPoints)
    sortedPoints = mergeSort(listOfPoints, axis)

    if n <= 3:
        d, point1, point2, counter = closestPairBF(sortedPoints)
    else:
        halfn = n//2

        if (n%2 == 0):
            space1 = sortedPoints[0:halfn]
            space2 = sortedPoints[halfn:n]
            middleLine = (sortedPoints[halfn-1][axis] +
sortedPoints[halfn][axis]) / 2
```

```
        else:
            space1 = sortedPoints[0:halfn]
            space2 = sortedPoints[halfn+1:n]
            middleLine = sortedPoints[halfn][axis]

        leftDist, leftPoint1, leftPoint2, counter = closestPairDNC(space1)
        rightdist, rightPoint1, rightPoint2, temp =  closestPairDNC(space2)
        counter += temp

        if (leftDist < rightdist):
            d = leftDist
            point1 = leftPoint1
            point2 = leftPoint2
        else:
            d = rightdist
            point1 = rightPoint1
            point2 = rightPoint2

        stripPoint = stripList(sortedPoints, middleLine, d)
        sortedStrip = mergeSort(stripPoint, axis+1)

        stripDist, stripPoint1, stripPoint2, temp = closestPairBF(sortedStrip,
filter=lambda point1, point2: ignore(point1, point2, d))
        counter += temp

        if (stripDist < d):
            d = stripDist
            point1 = stripPoint1
            point2 = stripPoint2


    return d, point1, point2, counter


#menentukan pasangan terdekat dengan Brute Force
def closestPairBF(listOfPoints, filter = None):
    nPoints = len(listOfPoints)
    nDim = len(listOfPoints[0])
```

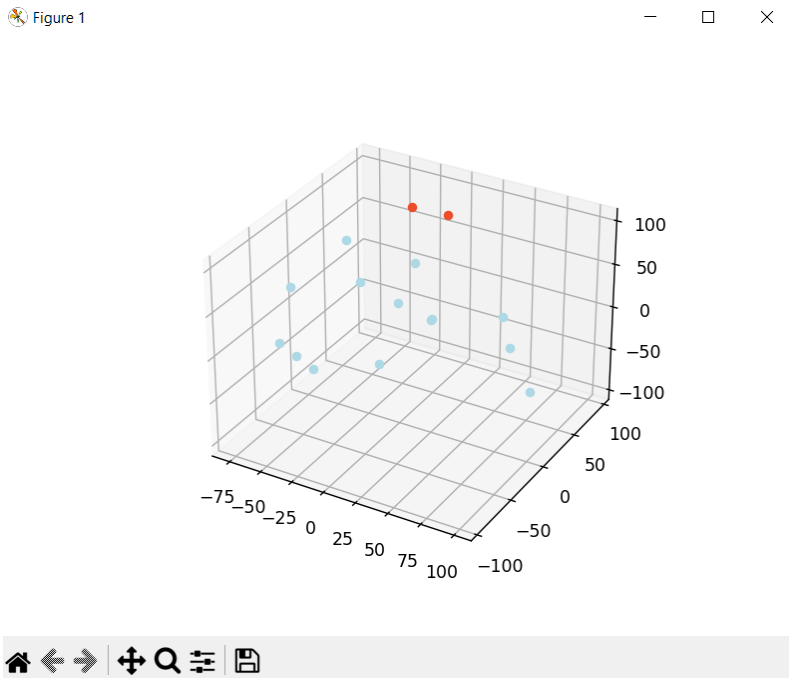
```
    counter = 0

    d = eucDistance(listOfPoints[0], listOfPoints[1], nDim)
    pair1 = []
    pair2 = []
    i = 0
    for i in range(nPoints):
        for j in range(i+1,nPoints):
            if(filter!=None and filter(listOfPoints[i], listOfPoints[j])):
                continue
            distResult = eucDistance(listOfPoints[i], listOfPoints[j], nDim)
            counter += 1
            if d > distResult:
                d = distResult
                pair1 = listOfPoints[i]
                pair2 = listOfPoints[j]

    return d, pair1, pair2, counter
```

BAB IV

TEST PROGRAM

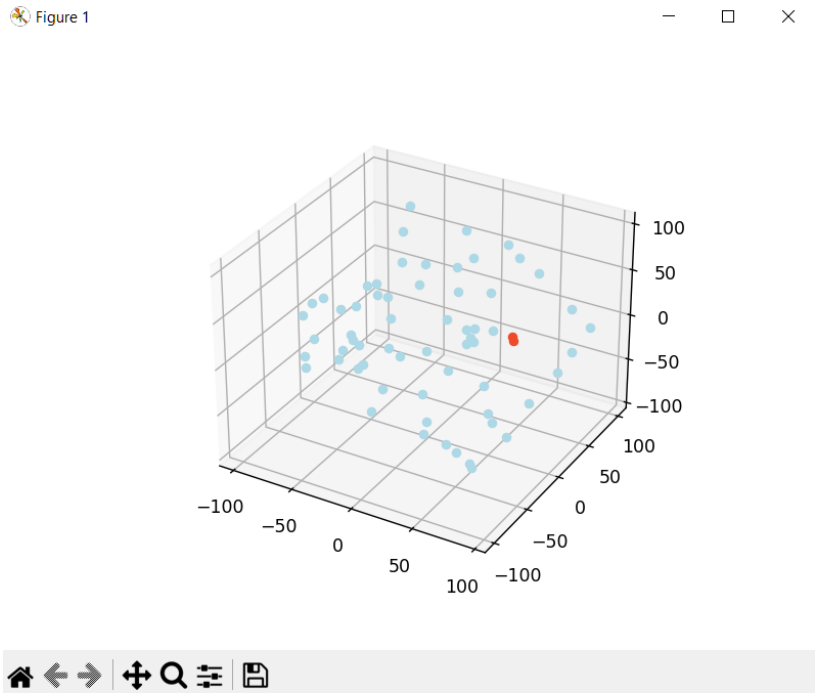
1. Testcase 1 (n=16)

```
Menu
1. 3 Dimensional Space
2. Euclidean Space (Vector Rn)
3. Keluar Program
Masukan pilihan menu: 1
Banyak point: 16
Closest pair dengan Brute Force:
Jarak terdekat 26.68475491020774
Titik 1: [0.9677369988041704, 70.35716045181562, 87.51742587028494]
Titik 2: [-25.412953308829643, 66.4048308693024, 88.23465273938726]
Waktu eksekusi (s): 0.0
Banyak operasi Euclidean Distance: 120
Running on Windows-10-10.0.19044-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel

Closest pair dengan Divide and Conquer:
Jarak terdekat 26.68475491020774
Titik 1: [-25.412953308829643, 66.4048308693024, 88.23465273938726]
Titik 2: [0.9677369988041704, 70.35716045181562, 87.51742587028494]
Waktu eksekusi (s): 0.0
Banyak operasi Euclidean Distance: 27
Running on Windows-10-10.0.19044-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```



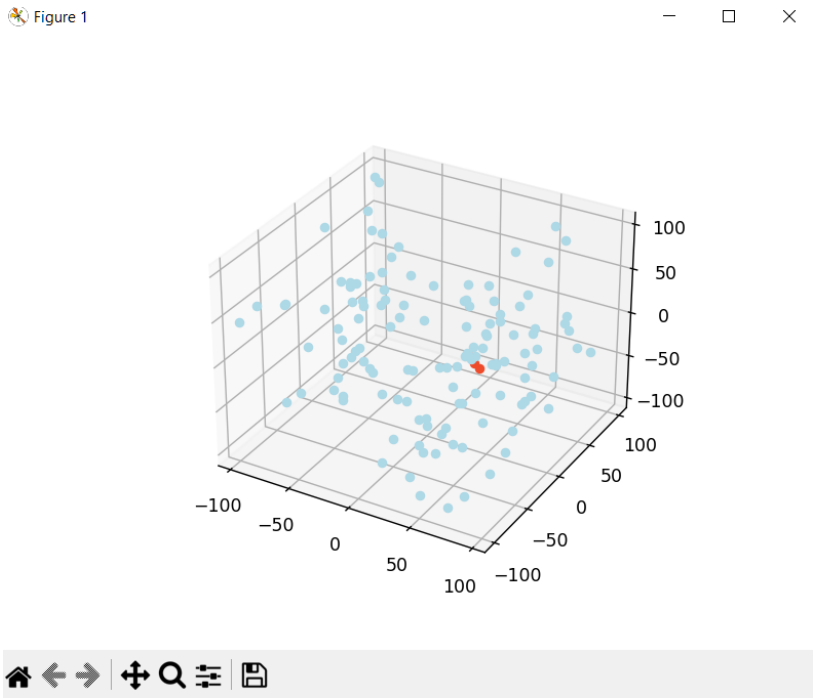
2. Testcase 2 (n=64)

```
Menu
1. 3 Dimensional Space
2. Euclidean Space (Vector Rn)
3. Keluar Program
Masukan pilihan menu: 1
Banyak point: 64
Closest pair dengan Brute Force:
Jarak terdekat 4.462426099331658
Titik 1: [74.04689861429452, -11.944969398186501, 42.965556545148786]
Titik 2: [76.88599537041975, -14.618225541902902, 40.79611470646353]
Waktu eksekusi (s): 0.0020012855529785156
Banyak operasi Euclidean Distance: 2016
Running on Windows-10-10.0.19044-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel

Closest pair dengan Divide and Conquer:
Jarak terdekat 4.462426099331658
Titik 1: [76.88599537041975, -14.618225541902902, 40.79611470646353]
Titik 2: [74.04689861429452, -11.944969398186501, 42.965556545148786]
Waktu eksekusi (s): 0.001999378204345703
Banyak operasi Euclidean Distance: 108
Running on Windows-10-10.0.19044-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```



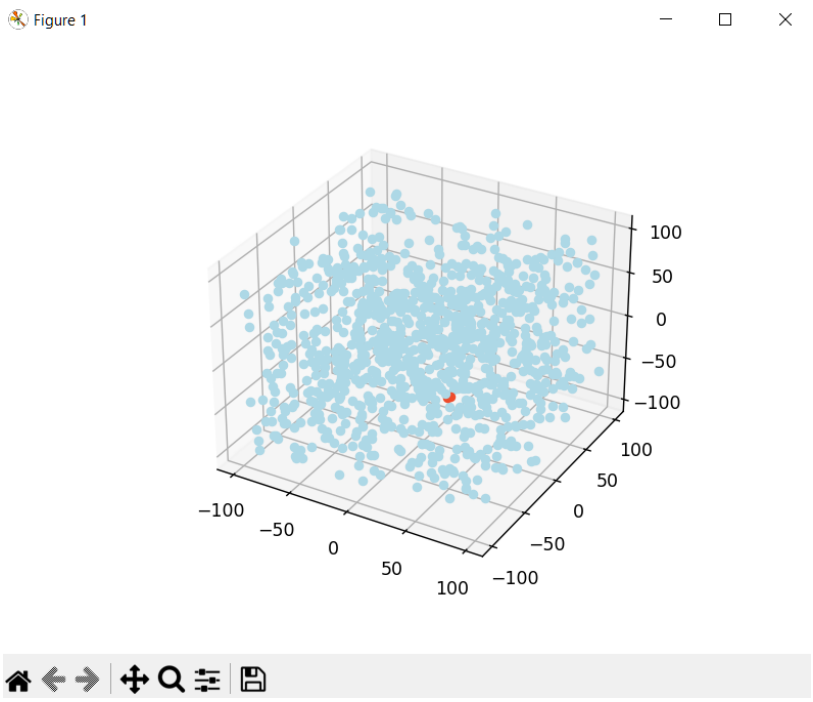
3. Testcase 3 (n=128)

```
Menu
1. 3 Dimensional Space
2. Euclidean Space (Vector Rn)
3. Keluar Program
Masukan pilihan menu: 1
Banyak point: 128
Closest pair dengan Brute Force:
Jarak terdekat 6.638444141909032
Titik 1: [-9.177029474162097, 90.53142179581127, -91.78833455697865]
Titik 2: [-5.277979294282417, 92.16892457866248, -96.90545610685228]
Waktu eksekusi (s): 0.009000539779663086
Banyak operasi Euclidean Distance: 8128
Running on Windows-10-10.0.19044-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel

Closest pair dengan Divide and Conquer:
Jarak terdekat 6.638444141909032
Titik 1: [-9.177029474162097, 90.53142179581127, -91.78833455697865]
Titik 2: [-5.277979294282417, 92.16892457866248, -96.90545610685228]
Waktu eksekusi (s): 0.004009723663330078
Banyak operasi Euclidean Distance: 205
Running on Windows-10-10.0.19044-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```



4. Testcase 4 (n=1000)

```
Menu
1. 3 Dimensional Space
2. Euclidean Space (Vector Rn)
3. Keluar Program
Masukan pilihan menu: 1
Banyak point: 1000
Closest pair dengan Brute Force:
Jarak terdekat 2.490016277802975
Titik 1: [21.31410005940006, 2.1898698558972427, -55.67635347793476]
Titik 2: [19.51340606263605, 0.4741858123823164, -55.795140920748]
Waktu eksekusi (s): 0.5060584545135498
Banyak operasi Euclidean Distance: 499500
Running on Windows-10-10.0.19044-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel

Closest pair dengan Divide and Conquer:
Jarak terdekat 2.490016277802975
Titik 1: [19.51340606263605, 0.4741858123823164, -55.795140920748]
Titik 2: [21.31410005940006, 2.1898698558972427, -55.67635347793476]
Waktu eksekusi (s): 0.0559999942779541
Banyak operasi Euclidean Distance: 1531
Running on Windows-10-10.0.19044-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```



5. Testcase Vector 1 (dimensi = 5, n=128)

```
Menu
1. 3 Dimensional Space
2. Euclidean Space (Vector Rn)
3. Keluar Program
Masukan pilihan menu: 2
Dimensi: 5
Banyak point: 128
Closest pair dengan Brute Force:
Jarak terdekat 22.66662011495586
Titik 1: [-26.871053938894192, -72.41578952115276, 85.4312612023551, -69.6638146478745, 52.46329604753561]
Titik 2: [-38.2360273153189, -87.21337299899913, 80.01292042565169, -59.17879151359053, 57.59656439295554]
Waktu eksekusi (s): 0.009511232376098633
Banyak operasi Euclidean Distance: 8128
Running on Windows-10-10.0.22621-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel

Closest pair dengan Divide and Conquer:
Jarak terdekat 22.66662011495586
Titik 1: [-38.2360273153189, -87.21337299899913, 80.01292042565169, -59.17879151359053, 57.59656439295554]
Titik 2: [-26.871053938894192, -72.41578952115276, 85.4312612023551, -69.6638146478745, 52.46329604753561]
Waktu eksekusi (s): 0.005999326705932617
Banyak operasi Euclidean Distance: 307
Running on Windows-10-10.0.22621-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```

6. Testcase Vector 2 (dimensi = 7, n=1000)

```
Menu
1. 3 Dimensional Space
2. Euclidean Space (Vector Rn)
3. Keluar Program
Masukan pilihan menu: 2
Dimensi: 7
Banyak point: 1000
Closest pair dengan Brute Force:
Jarak terdekat 27.888080554541524
Titik 1: [-87.10172641394314, 28.210475840710473, -15.44510001689008, -34.053780683797015, 62.118063211235125, -8.4372882777118, -15.843753829833503
]
Titik 2: [-98.60288710649321, 39.00887392288021, -0.902547040530763, -28.271559678362053, 71.34150959315798, -6.501277960596383, -29.812394159716632
]
Waktu eksekusi (s): 0.7930076122283936
Banyak operasi Euclidean Distance: 499500
Running on Windows-10-10.0.22621-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel

Closest pair dengan Divide and Conquer:
Jarak terdekat 27.888080554541524
Titik 1: [-87.10172641394314, 28.210475840710473, -15.44510001689008, -34.053780683797015, 62.118063211235125, -8.4372882777118, -15.843753829833503
]
Titik 2: [-98.60288710649321, 39.00887392288021, -0.902547040530763, -28.271559678362053, 71.34150959315798, -6.501277960596383, -29.812394159716632
]
Waktu eksekusi (s): 0.2140183448791504
Banyak operasi Euclidean Distance: 4233
Running on Windows-10-10.0.22621-SP0 Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```

LAMPIRAN

Link Github : https://github.com/ChrisAlberth/Tucil2_13521078_13521165.git

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. Bonus 1 dikerjakan	✓	
6. Bonus 2 dikerjakan	✓	