IF2211 STRATEGI ALGORITMA

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

LAPORAN TUGAS KECIL 2

Disusun oleh:
Kenny Benaya Nathan - 13521023

TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

DAFTAR ISI

BAB I
ALGORITMA *DIVIDE AND CONQUER*

1.1. Dasar Teori

Algoritma *Divide and Conquer* adalah salah satu algoritma yang cukup populer. Namanya sendiri diambil dari salah satu strategi militer yang cukup populer juga, yaitu *"Devide et Impera"*. Sesuai dengan asal namanya, algoritma ini membagi-bagi sebuah masalah yang ingin dipecahkan menjadi beberapa bagian sampai bagian kecil dari masalah tersebut tidak bisa dibagi-bagi lagi. Kemudian, setiap submasalah tersebut diselesaikan kemudian digabung dengan submasalah lain yang sudah diselesaikan sehingga menghasilkan sebuah solus dari 2 submasalah yang digabungi. Proses ini berjalan secara rekursif sehingga akan menghasilkan solusi dari masalah besar tadi.

1.2. Pencarian Pasangan Titik Dengan Jarak Terdekat

Jika diberikan sejumlah titik, pendekatan *Divide and Conquer* dapat digunakan untuk mencari sebuah pasangan dengan jarak terdekat dari antara pasangan titik lain pada ruang berdimensi 2. Jarak antara dua titik dapat dihitung dengan menggunakan rumus *Euclidean Distance*. Jika sedang dicari jarak antara titik a dan b yang memiliki n dimensi, maka rumus *Euclidean Distance* yang digunakan dapat dilihat pada persamaan (1).

$$d = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2} \tag{1}$$

BAB II
IMPLEMENTASI ALGORITMA

2.1. Pencarian Pasangan Titik Dengan Jarak Terdekat Dalam Ruang Dimensi 3

Pendekatan *Divide and Conquer* juga bisa dilakukan untuk mencari pasangan titik dengan jarak terdekat dalam ruang dimensi 3, bahkan bisa lebih dari 3. Algoritma yang digunakan untuk menyelesaikan masalah ini sebagai berikut:

1. Seluruh titik dikumpulkan dalam satu himpunan (misal: 'S') kemudian diurut berdasarkan salah satu garis (misal garis dimensi X)
2. Bagi dua himpunan S sehingga jumlah pada S1 sama dengan atau kurang satu dari himpunan S2. Jadi, jika n adalah jumlah titik, maka isi dari S1 adalah S[1..n/2] dan isi dari S2 adalah S[n/2+1..n]
3. Ulangi langkah ke-2 terus menerus sampai akhirnya jumlah titik pada himpunan paling kecil adalah 2 atau 3
4. Tentukan jarak paling kecil dari masing-masing sub himpunan dengan menggunakan *euclidean distance* seperti pada persamaan (2).

$$d = \sqrt{\Sigma_{i=1}^{n}(ai - bi)^2} \tag{2}$$

   a. Jika ada 2 titik, maka hitung saja jarak antara kedua titik tersebut
   b. Jika ada 3 titik, maka hitung jarak masing-masing pasangan secara *brute-force* sehingga ditemukan jarak paling kecil
5. Gabung masing-masing sub himpunan dengan menentukan jarak yang paling kecil dari kedua sub himpunan tersebut
6. Buatlah sebuah garis imajiner yang letaknya ada di garis dimensi pertama pada titik paling tengah
7. Kumpulkan titik yang jaraknya dengan garis imajiner kurang dari jarak paling kecil saat itu
8. Urutkan kumpulan titik tersebut berdasarkan garis dimensi berikutnya (misal garis dimensi Y)
9. Jika jarak antara suatu pasangan titik dari garis dimensi kedua kurang dari jarak paling kecil saat itu, maka hitung *euclidean distance* dari pasangan tersebut dan tentukan mana yang terkecil. Evaluasi semua pasangan titik pada himpunan tersebut

10. Ulangi langkah ke-4 sampai 9 untuk menggabungkan seluruh subhimpunan sehingga pada akhirnya menemukan pasangan dengan jarak terkecil dari satu himpunan S

2.2. *Source* Program

Pada program ini, pengembang membatasi letak koordinat titik, yaitu terletak di antara titik -1000 sampai 1000. Berikut *source* program yang dibuat.

2.2.1. src/main.py

```
import time
import platform

import plot as plot
import functions as func
import algorithms as algorithm

if __name__ == '__main__':
    # GLOBAL VARIABLE
    closestPair = []
    closestDistance = 0
    count = 0
    points = []

    # START MENU
    print('CLOSEST PAIR OF RANDOM POINTS IN 3D (AND MORE) SPACE')
    print('============================================================')

    # INPUT POINTS
    func.inputPoint(points)

    # DIVIDE AND CONQUER
    print('============================================================')
    print("DIVIDE AND CONQUER")
    print("------------------")
    start_dnc = time.time()
    func.sort(points, 'x0')
    closestDistance, closestPair, count = algorithm.divideConquer(points, count)
    end_dnc = time.time()
    print("Shortest Distance                = " + str(closestDistance))
    print("Euclidean Distance Calculated    = " + str(count))
    print("Time taken                       = {:.2f} ms".format((end_dnc -
start_dnc) * 1000))
    print("Closest Pair(s)")
    func.printPairs(closestPair)
    plot.plot(points, closestPair)
```

```
    _ = input('Press Enter to continue and see Brute Force Algorithm')

    # BRUTE FORCE
    print('============================================================')
    print("BRUTE FORCE")
    print("-----------")
    closestPair = []
    closestDistance = 0
    count = 0
    start_bf = time.time()
    closestDistance, closestPair, count = algorithm.bruteForce(points, count)
    end_bf = time.time()
    print("Shortest Distance                = " + str(closestDistance))
    print("Euclidean Distance Calculated    = " + str(count))
    print("Time taken                       = {:.2f} ms".format((end_bf -
start_bf) * 1000))
    print("Closest Pair(s)")
    func.printPairs(closestPair)
    plot.plot(points, closestPair)
    print('============================================================')
    print('Device used: ' + platform.processor())
```

2.2.2. *src/algorithms.py*

```
import functions as func
# BRUTE FORCE ALGORITHM
def bruteForce(points, count):

    tempClosestPair = [[points[0], points[1]]]
    min_distance, count = func.distance(points[0], points[1], count)

    for i in range(len(points)):
        for j in range(i+1, len(points)):
            if not (i == 0 and j == 1):
                temp_distance, count = func.distance(points[i], points[j], count)

                if temp_distance < min_distance:
                    min_distance = temp_distance
                    tempClosestPair = [[points[i], points[j]]]
                elif temp_distance == min_distance:
                    if ([points[i], points[j]] not in tempClosestPair) and
([points[j], points[i]] not in tempClosestPair):
                        tempClosestPair.append([points[i], points[j]])

    return min_distance, tempClosestPair, count
```

```python
# DIVIDE AND CONQUER ALGORITHM
def divideConquer(points, count):
    tempClosestPair = []

    # BASE
    if len(points) == 2:
        min_distance, count = func.distance(points[0], points[1], count)
        tempClosestPair = [[points[0], points[1]]]
        return min_distance, tempClosestPair, count

    elif len(points) == 3:
        return bruteForce(points, count)

    # DIVIDE
    mid = len(points)//2
    left = points[:mid]
    right = points[mid:]

    # RECURSIVE
    min_left, pair_left, count = divideConquer(left, count)
    min_right, pair_right, count = divideConquer(right, count)

    if min_left == min_right:
        tempClosestPair = pair_right + pair_left
        min_distance = min_left
    else:
        tempClosestPair = pair_left if min_left < min_right else pair_right
        min_distance = min_left if min_left < min_right else min_right

    mid_line = points[mid]['x0']
    strip = []
    for i in range(len(points)):
        if abs(points[i]['x0'] - mid_line) < min_distance:
            strip.append(points[i])
    if len(points[0]) != 1:
        func.sort(strip, 'x1')

    # CONQUER
    for i in range(len(strip)):
        for j in range(i+1, len(strip)):
            for k in range(len(strip[i])):
                if abs(strip[j]['x' + str(k)] - strip[i]['x' + str(k)]) <
min_distance:
                    dis_temp, count = func.distance(strip[i], strip[j], count)
                    if dis_temp < min_distance:
                        min_distance = dis_temp
                        tempClosestPair = [[strip[i], strip[j]]]
```

```
                    elif dis_temp == min_distance:
                        if ([strip[i], strip[j]] not in tempClosestPair) and
([strip[j], strip[i]] not in tempClosestPair):
                            tempClosestPair.append([strip[i], strip[j]])

    return min_distance, tempClosestPair, count
```

2.2.3. *src/functions.py*

```
import math
import random

# CALCULATE EUCLIDEAN DISTANCE BETWEEN TWO POINTS
def distance(p1, p2, count):
    count += 1
    dist = 0
    for i in range(len(p1)):
        dist += (p2['x' + str(i)] - p1['x' + str(i)])**2
    return math.sqrt(dist), count

# SORT POINTS BY (AXIS)
def sort(points, axis):
    points.sort(key=lambda x: x[axis])

def printPairs(pairs):
    for i in range (len(pairs)):
        print(i + 1, end = '. ')
        for j in range (2):
            for k in range(len(pairs[i][0]) - 1):
                if k == 0:
                    print("({:.2f}, ".format(pairs[i][j]['x' + str(k)]), end = '')
                else:
                    print("{:.2f}, ".format(pairs[i][j]['x' + str(k)]), end = '')

            if j == 0:
                print("{:.2f})".format(pairs[i][j]['x' + str(len(pairs[i][0]) -
1)]), end = ' and ')
            else:
                print("{:.2f})".format(pairs[i][j]['x' + str(len(pairs[i][0]) -
1)]))

# INPUT POINTS
def inputPoint(points):
    flag = False
```

```
    # INPUT AMOUNT OF POINTS
    while(not flag):
        try:
            p = int(input("Input amount of point(s): "))
            if p < 2:
                print("Invalid input! (Please input integer more than 1")
            else:
                flag = True
        except ValueError:
            print("ERROR! (Input must be an integer and more than 1)")

    # INPUT DIMENSIONAL SPACE
    flag = False
    while(not flag):
        try:
            d = int(input("Input dimensional space: "))
            if d < 1:
                print("Invalid input! (Please input integer more than 1)")
            else:
                flag = True
        except ValueError:
            print("ERROR! (Input must be integer and more than 1)")

    # GENERATE RANDOM COORDINATES
    while(len(points) < p):
        coordinate = {}
        for i in range(d):
            coordinate['x' + str(i)] = random.uniform(-1000,1000)
        if coordinate not in points:
            points.append(coordinate)
    return points
```

2.2.4. src/plot.py

```
import matplotlib.pyplot as plt

def plot(points, closestPair):
    # INITIALIZE ARRAYS
    x1 = []
    x2 = []
    x3 = []
    x4 = []
    x5 = []
    x6 = []
    x_line = []
    y_line = []
    z_line = []
```

```
    # SEPARATE POINTS INTO ARRAYS OF EACH DIMENSION
    for i in range(len(points)):
        if len(points[i]) >= 1:
            x1.append(points[i]['x0'])
        if len(points[i]) >= 2:
            x2.append(points[i]['x1'])
        if len(points[i]) >= 3:
            x3.append(points[i]['x2'])
        if len(points[i]) >= 4:
            x4.append(points[i]['x3'])
        if len(points[i]) >= 5:
            x5.append(points[i]['x4'])
        if len(points[i]) >= 6:
            x6.append(points[i]['x5'] * 1/10)

    # PLOT
    fig = plt.figure()
    if len(points[0]) <= 2:

        # 1 DIMENSION
        if len(points[0]) == 1:
            plt.scatter(x1, [0]*len(x1), color='black', alpha = 1)
            for i in range(len(closestPair)):
                plt.scatter(closestPair[i][0]['x0'], 0, color='red')
                plt.scatter(closestPair[i][1]['x0'], 0, color='red')
                x_line.append([closestPair[i][0]['x0'], closestPair[i][1]['x0']])
                y_line.append([0, 0])

        # 2 DIMENSIONS
        elif len(points[0]) == 2:
            plt.scatter(x1, x2, color='black', alpha = 1)
            for i in range(len(closestPair)):
                plt.scatter(closestPair[i][0]['x0'], closestPair[i][0]['x1'],
color='red')
                plt.scatter(closestPair[i][1]['x0'], closestPair[i][1]['x1'],
color='red')
                x_line.append([closestPair[i][0]['x0'], closestPair[i][1]['x0']])
                y_line.append([closestPair[i][0]['x1'], closestPair[i][1]['x1']])

        # PLOT LINES
        for i in range(len(x_line)):
            plt.plot(x_line[i], y_line[i])
        plt.show()

    elif len(points[0]) <= 6:
        space = fig.add_subplot(111, projection='3d')
```

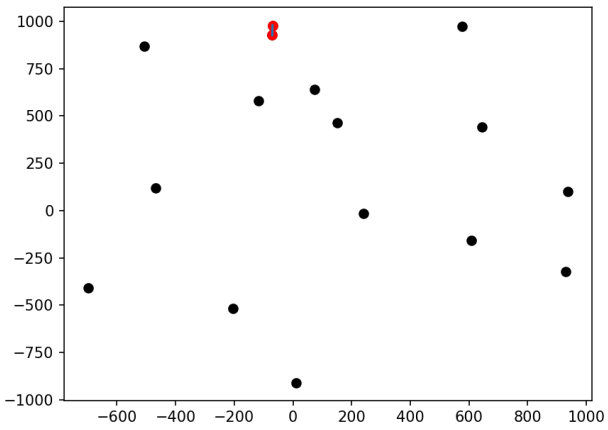
```
        # 3 DIMENSIONS
        if len(points[0]) == 3:
              space.scatter(x1, x2, x3, color='black', alpha = 0.5)
              for i in range(len(closestPair)):
                space.scatter(closestPair[i][0]['x0'], closestPair[i][0]['x1'],
closestPair[i][0]['x2'], color='red')
                space.scatter(closestPair[i][1]['x0'], closestPair[i][1]['x1'],
closestPair[i][1]['x2'], color='red')

        # 4 DIMENSIONS
        elif len(points[0]) == 4:
            img = space.scatter(x1, x2, x3, c=x4, cmap=plt.hot(), alpha = 1)
            fig.colorbar(img)

        # 5 DIMENSIONS
        elif len(points[0]) == 5:
            img = space.scatter(x1, x2, x3, c=x4, cmap=plt.hot(), s = x5, alpha =
1)
            fig.colorbar(img)

        # 6 DIMENSIONS
        elif len(points[0]) == 6:
            img = space.scatter(x1, x2, x3, c=x4, cmap=plt.hot(), s = x5, alpha =
x6)
            fig.colorbar(img)

        # PLOT LINES
        for i in range(len(closestPair)):
            x_line.append([closestPair[i][0]['x0'], closestPair[i][1]['x0']])
            y_line.append([closestPair[i][0]['x1'], closestPair[i][1]['x1']])
            z_line.append([closestPair[i][0]['x2'], closestPair[i][1]['x2']])
            space.plot(x_line[i], y_line[i], z_line[i])

        space.set_xlabel('X Label')
        space.set_ylabel('Y Label')
        space.set_zlabel('Z Label')
        plt.show()

    else:
        print("Too many dimensions to plot!")
```

BAB III
HASIL PENGUJIAN PROGRAM

3.1. Uji Dimensi 2

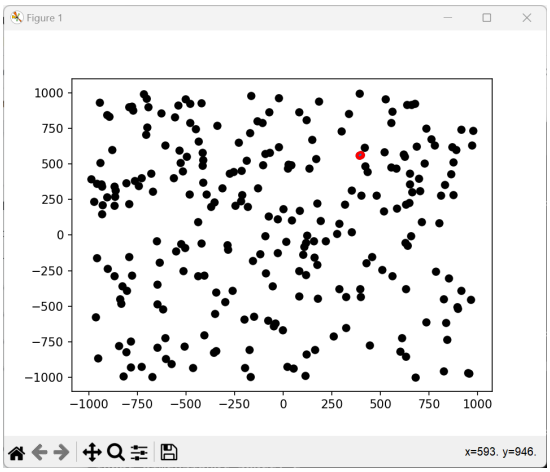
3.1.1. Uji 16 Titik

```
CLOSEST PAIR OF RANDOM POINTS IN 3D (AND MORE) SPACE
============================================================
Input amount of point(s): 16
Input dimensional space: 2
============================================================
DIVIDE AND CONQUER
------------------
Shortest Distance               = 51.938115552300886
Euclidean Distance Calculated   = 43
Time taken                      = 0.00 ms
Closest Pair(s)
1. (-70.30, 927.14) and (-69.54, 979.08)
Press Enter to continue and see Brute Force Algorithm
============================================================
BRUTE FORCE
-----------
Shortest Distance               = 51.938115552300886
Euclidean Distance Calculated   = 120
Time taken                      = 0.00 ms
Closest Pair(s)
1. (-70.30, 927.14) and (-69.54, 979.08)
============================================================
Device used: Intel64 Family 6 Model 154 Stepping 3, GenuineIntel
```



3.1.2. Uji 256 Titik

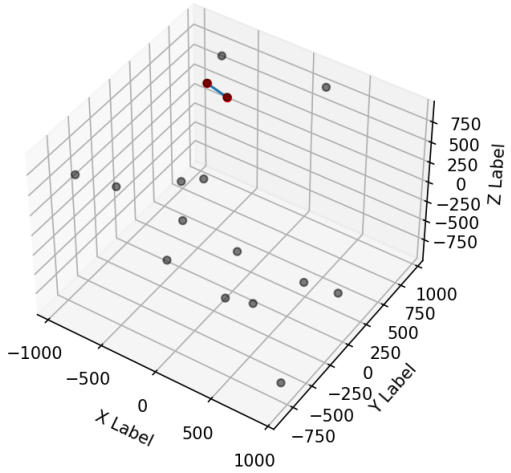
```
Windows PowerShell
Device used: Intel64 Family 6 Model 154 Stepping 3, GenuineIntel
PS C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space> ./run

C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space>python -X pycache_prefix=bin src/main.py
CLOSEST PAIR OF RANDOM POINTS IN 3D (AND MORE) SPACE
============================================================
Input amount of point(s): 256
Input dimensional space: 2
============================================================
DIVIDE AND CONQUER
------------------
Shortest Distance               = 6.80538928984931
Euclidean Distance Calculated   = 1796
Time taken                      = 4.71 ms
Closest Pair(s)
1. (393.78, 560.90) and (398.39, 565.91)
Press Enter to continue and see Brute Force Algorithm
============================================================
BRUTE FORCE
-----------
Shortest Distance               = 6.80538928984931
Euclidean Distance Calculated   = 32640
Time taken                      = 56.36 ms
Closest Pair(s)
1. (393.78, 560.90) and (398.39, 565.91)
============================================================
Device used: Intel64 Family 6 Model 154 Stepping 3, GenuineIntel
PS C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space>
```



3.2. Uji Dimensi 3

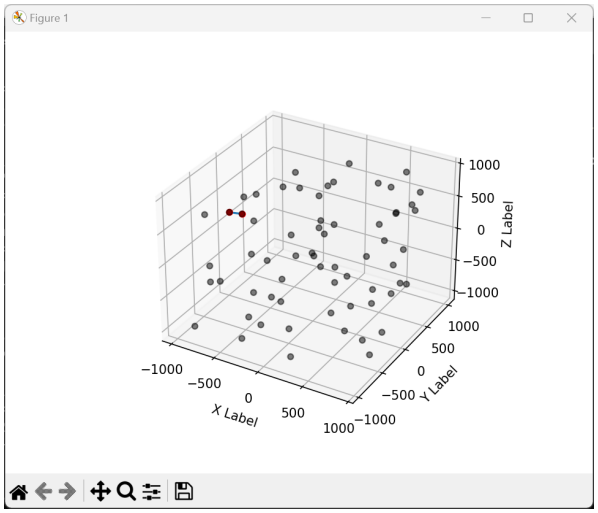
3.2.1. Uji 16 Titik

```
Windows PowerShell
Device used: Intel64 Family 6 Model 154 Stepping 3, GenuineIntel
PS C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space> ./run

C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space>python -X pycache_prefix=bin src/main.py
CLOSEST PAIR OF RANDOM POINTS IN 3D (AND MORE) SPACE
============================================================
Input amount of point(s): 16
Input dimensional space: 3
============================================================
DIVIDE AND CONQUER
------------------
Shortest Distance               = 250.42840982892147
Euclidean Distance Calculated   = 129
Time taken                      = 1.63 ms
Closest Pair(s)
1. (−519.20, 680.98, 505.08) and (−757.39, 754.32, 480.57)
Press Enter to continue and see Brute Force Algorithm
============================================================
BRUTE FORCE
-----------
Shortest Distance               = 250.42840982892147
Euclidean Distance Calculated   = 120
Time taken                      = 0.00 ms
Closest Pair(s)
1. (−757.39, 754.32, 480.57) and (−519.20, 680.98, 505.08)
============================================================
Device used: Intel64 Family 6 Model 154 Stepping 3, GenuineIntel
PS C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space>
```

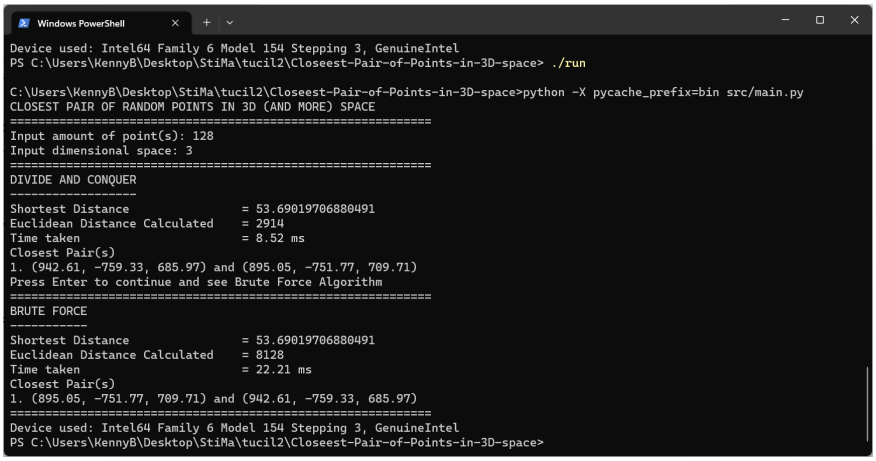


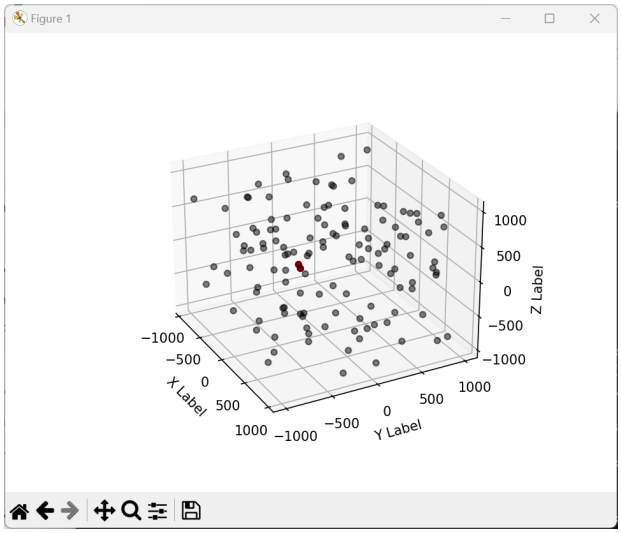
3.2.2. Uji 64 Titik

```
Windows PowerShell
Device used: Intel64 Family 6 Model 154 Stepping 3, GenuineIntel
PS C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space> ./run

C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space>python -X pycache_prefix=bin src/main.py
CLOSEST PAIR OF RANDOM POINTS IN 3D (AND MORE) SPACE
============================================================
Input amount of point(s): 64
Input dimensional space: 3
============================================================
DIVIDE AND CONQUER
------------------
Shortest Distance               = 161.99986969948758
Euclidean Distance Calculated   = 1197
Time taken                      = 4.18 ms
Closest Pair(s)
1. (-581.65, -628.29, 736.19) and (-514.84, -502.91, 658.34)
Press Enter to continue and see Brute Force Algorithm
============================================================
BRUTE FORCE
-----------
Shortest Distance               = 161.99986969948758
Euclidean Distance Calculated   = 2016
Time taken                      = 3.53 ms
Closest Pair(s)
1. (-581.65, -628.29, 736.19) and (-514.84, -502.91, 658.34)
============================================================
Device used: Intel64 Family 6 Model 154 Stepping 3, GenuineIntel
PS C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space>
```

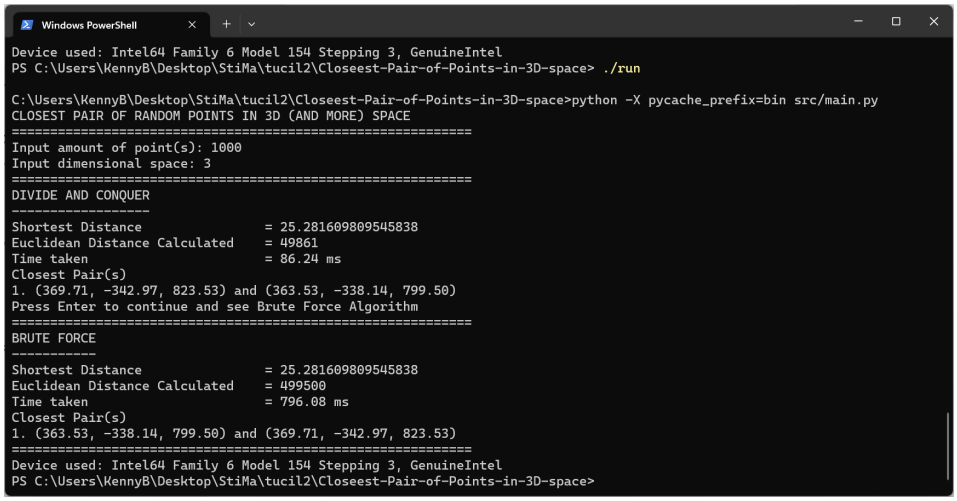


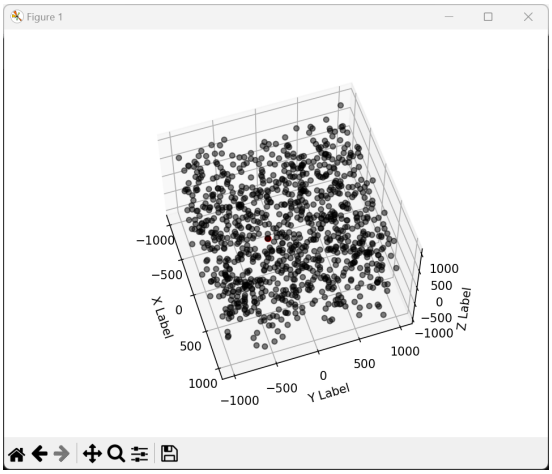
3.2.3. Uji 128 Titik





3.2.4. Uji 1000 Titik

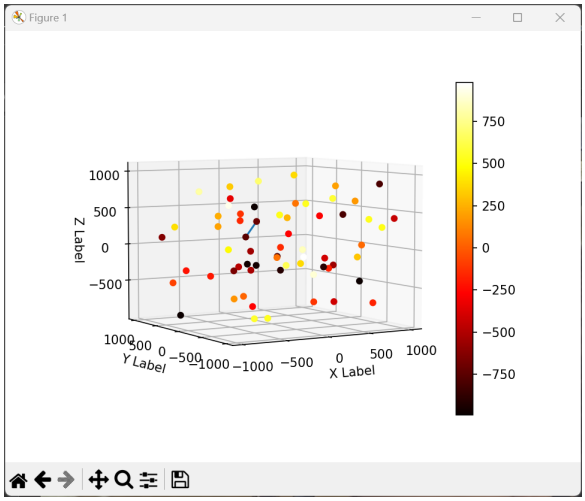




3.3. Uji Dimensi 4

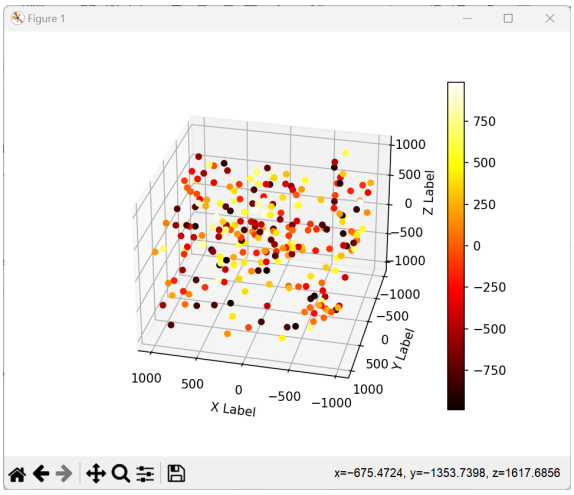
3.3.1. Uji 64 Titik

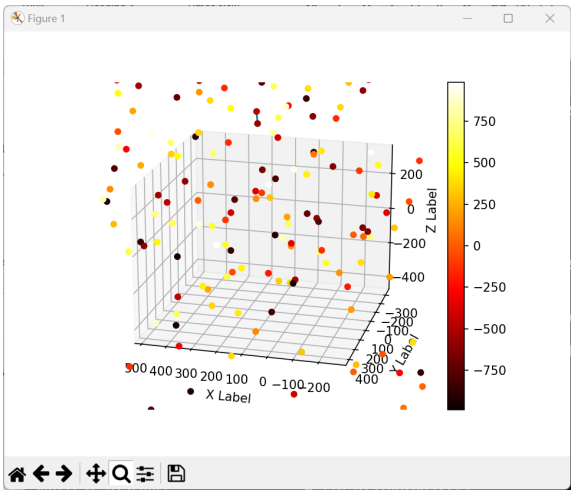
```
Windows PowerShell
Device used: Intel64 Family 6 Model 154 Stepping 3, GenuineIntel
PS C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space> ./run

C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space>python -X pycache_prefix=bin src/main.py
CLOSEST PAIR OF RANDOM POINTS IN 3D (AND MORE) SPACE
============================================================
Input amount of point(s): 64
Input dimensional space: 4
============================================================
DIVIDE AND CONQUER
------------------
Shortest Distance               = 231.53681786818518
Euclidean Distance Calculated   = 2122
Time taken                      = 6.83 ms
Closest Pair(s)
1. (-611.64, -731.37, 444.92, -708.45) and (-705.43, -676.34, 245.36, -752.73)
Press Enter to continue and see Brute Force Algorithm
============================================================
BRUTE FORCE
-----------
Shortest Distance               = 231.53681786818518
Euclidean Distance Calculated   = 2016
Time taken                      = 4.99 ms
Closest Pair(s)
1. (-705.43, -676.34, 245.36, -752.73) and (-611.64, -731.37, 444.92, -708.45)
============================================================
Device used: Intel64 Family 6 Model 154 Stepping 3, GenuineIntel
PS C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space>
```



3.3.2. *Uji 256 Titik*

```
Windows PowerShell
Device used: Intel64 Family 6 Model 154 Stepping 3, GenuineIntel
PS C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space> ./run

C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space>python -X pycache_prefix=bin src/main.py
CLOSEST PAIR OF RANDOM POINTS IN 3D (AND MORE) SPACE
============================================================
Input amount of point(s): 256
Input dimensional space: 4
============================================================
DIVIDE AND CONQUER
------------------
Shortest Distance               = 115.51910526037773
Euclidean Distance Calculated   = 19794
Time taken                      = 39.57 ms
Closest Pair(s)
1. (-81.64, 903.05, 972.90, -491.77) and (-87.20, 911.57, 920.53, -594.23)
Press Enter to continue and see Brute Force Algorithm
============================================================
BRUTE FORCE
-----------
Shortest Distance               = 115.51910526037773
Euclidean Distance Calculated   = 32640
Time taken                      = 64.65 ms
Closest Pair(s)
1. (-87.20, 911.57, 920.53, -594.23) and (-81.64, 903.05, 972.90, -491.77)
============================================================
Device used: Intel64 Family 6 Model 154 Stepping 3, GenuineIntel
PS C:\Users\KennyB\Desktop\StiMa\tucil2\Closeest-Pair-of-Points-in-3D-space>
```





LAMPIRAN

Tautan *Repository* : https://github.com/kennypanjaitan/Tucil2_13521023

Tabel *Checklist Goal*

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	V	
2	Program berhasil running	V	
3	Program dapat menerima masukan dan dan menuliskan luaran.	V	
4	Luaran program sudah benar (solusi closest pair benar)	V	
5	Bonus 1 dikerjakan	V	
6	Bonus 2 dikerjakan	V	