Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

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

Bernardus Willson
K03 / 13521021

1. Algoritma Divide and Conquer

Algoritma *Divide and Conquer* secara singkat memiliki prinsip memecah-mecah masalah yang ada menjadi beberapa bagian kecil sehingga lebih mudah untuk diselesaikan. Langkah-langkah umum algoritma *Divide and Conquer* adalah: 1. *Divide*: Membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama); 2. *Conquer*: Memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif); 3. *Combine*: Menggabungkan solusi masing-masing masalah sehingga membentuk solusi masalah semula.

```python
# calculate the distance between two points and count the number of distance calculation
def distance(p1, p2):
    global count
    count += 1
    dis = 0
    for i in range(len(p1)):
        dis += (p1[i] - p2[i]) ** 2
    return math.sqrt(dis)


def sort(points):
    def quickSort(points, start, end):
        if start < end:
            pivot = partition(points, start, end)
            quickSort(points, start, pivot - 1)
            quickSort(points, pivot + 1, end)

    def partition(points, start, end):
        pivot = points[end][0]
        i = start - 1
        for j in range(start, end):
            if points[j][0] <= pivot:
                i += 1
                points[i], points[j] = points[j], points[i]
        points[i + 1], points[end] = points[end], points[i + 1]
        return i + 1

    quickSort(points, 0, len(points) - 1)
    return points
```

```python
#find the shortest distance between two points
def shortestDistance(points):
    #even number of points base
    if len(points) == 2:
        return distance(points[0], points[1]), points[0], points[1]

    #odd number of points base
    elif len(points) == 3:
        minDistance = rand*rand
        point1 = points[0]
        point2 = points[1]
        #brute force comparison
        for i in range(len(points)):
            for j in range(i+1, len(points)):
                dis = distance(points[i], points[j])
                if dis < minDistance:
                    minDistance = dis
                    point1 = points[i]
                    point2 = points[j]
        return minDistance, point1, point2

    #recursive
    else:
        mid = len(points) // 2
        leftMinDistance, leftPoint1, leftPoint2 = shortestDistance(points[:mid])
        rightMinDistance, rightPoint1, rightPoint2 = shortestDistance(points[mid:])
        if leftMinDistance < rightMinDistance:
            minDistance = leftMinDistance
            point1 = leftPoint1
            point2 = leftPoint2
        else:
            minDistance = rightMinDistance
            point1 = rightPoint1
            point2 = rightPoint2

        #find the points that are close to the mid point
        midX = points[mid][0]
        minPoints = []
        for point in points:
            if abs(point[0] - midX) < minDistance:
                minPoints.append(point)


        #brute force comparison
        for i in range(len(minPoints)):
            for j in range(i+1, len(minPoints)):
                dis = distance(minPoints[i], minPoints[j])
                if dis < minDistance:
                    minDistance = dis
```

```
                point1 = minPoints[i]
                point2 = minPoints[j]


    return minDistance, point1, point2
```

Algoritma di atas terletak pada file main.py dimana algoritma tersebut dapat mencari jarak terdekat antara dua titik di dalam ruang N dimensi. Pertama, algoritma ini akan memeriksa jumlah titik yang diberikan, jika hanya terdiri dari 2 titik maka jarak antara kedua titik tersebut akan langsung dihitung menggunakan rumus jarak, dengan kata lain basis genap. Namun, jika terdapat 3 titik, algoritma akan melakukan perbandingan jarak antara ketiga titik menggunakan metode brute force karena kita tidak dapat membandingkan titik dengan jumlah ganjil, dengan kata lain basis ganjil.

Jika terdapat lebih dari 3 titik, algoritma akan membagi titik-titik tersebut menjadi dua bagian yang sama besar hingga hanya terdapat dua titik atau tiga titik pada masing-masing bagian. Lalu, akan dicari jarak terdekat pada setiap bagian secara rekursif dengan memanggil fungsi shortestDistance() kembali pada setiap bagian tersebut. Kemudian, jarak terdekat pada kedua bagian akan dibandingkan dan titik-titik yang menjadi pasangan jarak terdekat tersebut akan ditentukan.

Setelah itu, algoritma akan mencari jarak terdekat antara titik-titik yang berada di dekat bidang pemisah kedua bagian, yaitu dengan mengumpulkan titik-titik pada suatu daerah yang memiliki jarak sekitar minDistance. Kemudian, jarak antara setiap pasangan titik pada daerah tersebut akan dibandingkan menggunakan metode *brute force*. Apabila ditemukan jarak terdekat yang lebih kecil daripada jarak terdekat sebelumnya, maka pasangan titik yang baru tersebut akan ditentukan sebagai pasangan jarak terdekat.

Dalam setiap perbandingan jarak antar titik, algoritma menggunakan fungsi distance() untuk menghitung jarak antar dua titik di dalam ruang N dimensi. Fungsi ini juga menghitung jumlah perbandingan jarak yang dilakukan dengan menambahkan satu pada variabel global count setiap kali fungsi distance() dipanggil. Variabel count ini akan menyimpan total perbandingan jarak yang dilakukan oleh algoritma dari awal hingga akhir.

Selain itu terdapat juga penggunaan kode sorting (metode sorting yang digunakan adalah quick sort) pada algoritma, namun pemanggilan fungsi dilakukan di luar fungsi shortestDistance() karena proses sorting hanya perlu dilakukan sekali yaitu saat sebelum proses perhitungan dilakukan. Hal ini berguna untuk menghandle kasus ketika pemisah (garis vertikal yang membagi titik-titik menjadi dua bagian) berada tepat di suatu titik.

Dalam kasus seperti ini jika tidak digunakan sorting, algoritma akan memilih titik-titik pada kedua sisi pemisah yang tidak berurutan dalam list input. Hal ini akan membuat algoritma gagal mencari jarak terpendek yang benar karena ada kemungkinan titik-titik yang benar-benar berdekatan ditempatkan pada bagian yang berbeda dari pemisah dan diabaikan oleh algoritma.

Dengan pengurutan titik-titik berdasarkan koordinat x, kita dapat memastikan bahwa algoritma memilih titik-titik yang benar-benar berdekatan dan berada pada sisi yang sama dari pemisah saat melakukan pembandingan dan menghitung jarak terpendeknya. Oleh karena itu, sorting sangat penting dan diperlukan untuk memastikan keakuratan algoritma dalam menemukan jarak terpendek antar titik.

2. Source Code Program

2.1 **main.py**

```
import math
import time
import random
import others.plot as plot
import platform


# calculate the distance between two points and count the number of distance calculation
def distance(p1, p2):
    global count
    count += 1
    dis = 0
    for i in range(len(p1)):
        dis += (p1[i] - p2[i]) ** 2
    return math.sqrt(dis)


# quick sort the points based on x coordinate
def sort(points):
    def quickSort(points, start, end):
        if start < end:
            pivot = partition(points, start, end)
            quickSort(points, start, pivot - 1)
            quickSort(points, pivot + 1, end)

    def partition(points, start, end):
        pivot = points[end][0]
        i = start - 1
        for j in range(start, end):
            if points[j][0] <= pivot:
                i += 1
                points[i], points[j] = points[j], points[i]
        points[i + 1], points[end] = points[end], points[i + 1]
        return i + 1

    quickSort(points, 0, len(points) - 1)
    return points
```

```python
#find the shortest distance between two points
def shortestDistance(points):
    #even number of points base
    if len(points) == 2:
        return distance(points[0], points[1]), points[0], points[1]

    #odd number of points base
    elif len(points) == 3:
        minDistance = rand*rand
        point1 = points[0]
        point2 = points[1]
        #brute force comparison
        for i in range(len(points)):
            for j in range(i+1, len(points)):
                dis = distance(points[i], points[j])
                if dis < minDistance:
                    minDistance = dis
                    point1 = points[i]
                    point2 = points[j]
        return minDistance, point1, point2

    #recursive
    else:
        mid = len(points) // 2
        leftMinDistance, leftPoint1, leftPoint2 = shortestDistance(points[:mid])
        rightMinDistance, rightPoint1, rightPoint2 = shortestDistance(points[mid:])
        if leftMinDistance < rightMinDistance:
            minDistance = leftMinDistance
            point1 = leftPoint1
            point2 = leftPoint2
        else:
            minDistance = rightMinDistance
            point1 = rightPoint1
            point2 = rightPoint2

        #find the points that are close to the mid point
        midX = points[mid][0]
        minPoints = []
        for point in points:
            if abs(point[0] - midX) < minDistance:
                minPoints.append(point)


        #brute force comparison
        for i in range(len(minPoints)):
            for j in range(i+1, len(minPoints)):
                dis = distance(minPoints[i], minPoints[j])
                if dis < minDistance:
                    minDistance = dis
```

```
                    point1 = minPoints[i]
                    point2 = minPoints[j]

        return minDistance, point1, point2


def main():
    global count
    global rand
    # Open the text file in read mode
    with open('src/others/splashScreen.txt', 'r') as file:
        contents = file.read()
        print(contents)

    #input validations
    flag = False
    while not flag:
        n = int(input('Masukkan jumlah titik: '))
        if n < 2:
            print('Jumlah titik minimal 2! Silahkan masukkan kembali.\n')
        else:
            flag = True
    flag = False
    while not flag:
        d = int(input('Masukkan jumlah dimensi: '))
        if d < 1:
            print('Jumlah dimensi minimal 1! Silahkan masukkan kembali.\n')
        else:
            flag = True

    #generate random points
    points = []
    for i in range(n):
        point = []
        for j in range(d):
            point.append(random.uniform(0, rand))
        points.append(point)

    start = time.time()

    #sort the points based on x coordinate
    points = sort(points)

    #calculate the shortest distance
    minDistance, point1, point2 = shortestDistance(points)
    end = time.time()
    print('------------------------------------------------------------------------------------
---------')
    print('Dua titik yang paling berdekatan:')
```

```
    print('Titik 1:',', '.join("{:.2f}".format(p) for p in point1))
    print('Titik 2:',', '.join("{:.2f}".format(p) for p in point2))
    print('\nJaraknya adalah: {:.2f}'.format(minDistance))
    print('Banyaknya operasi perhitungan rumus Euclidian: ', count)
    print('-------------------------------------------------------------------------------------
---------')
    print('Waktu eksekusi: {:.2f} ms'.format((end - start) * 1000))
    print(platform.processor())

    #plot the points
    plot.plot(points, point1, point2, rand)


if __name__ == '__main__':
    count = 0
    rand = 1000.0
    main()
```

2.2 **plot.py**

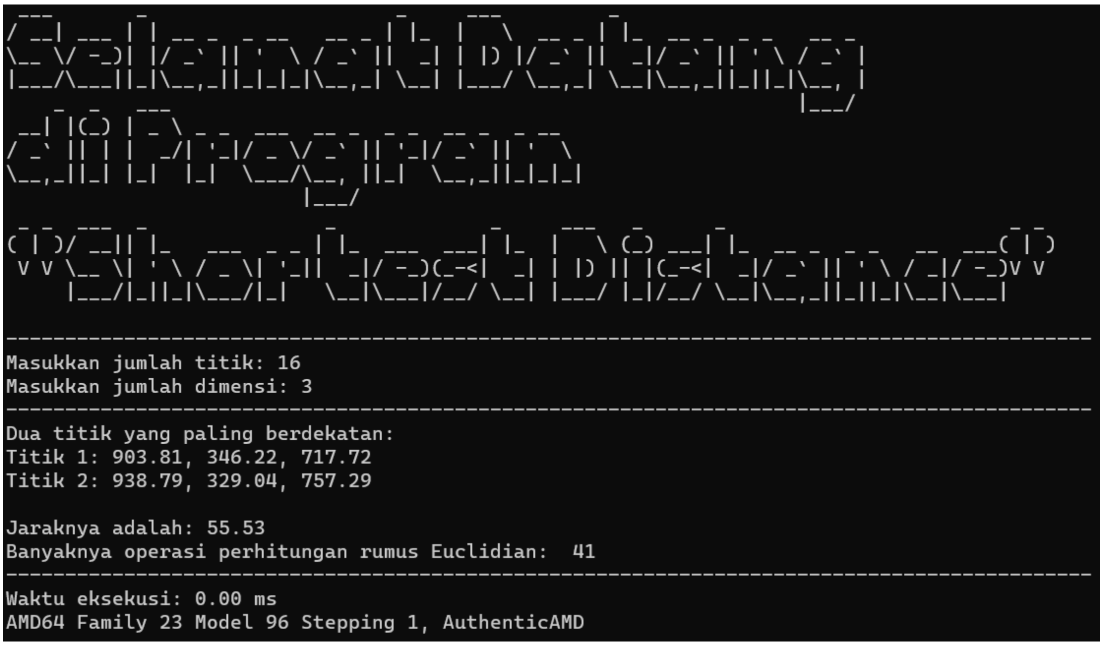
```
import matplotlib.pyplot as plt

#plot the points
def plot(points, point1, point2, rand):
    x = []
    y = []
    z = []
    c = []
    s = []
    o = []
    for point in points:
        if len(point) >= 1:
            x.append(point[0])
        if len(point) >= 2:
            y.append(point[1])
        if len(point) >= 3:
            z.append(point[2])
        if len(point) >= 4:
            c.append(point[3])
        if len(point) >= 5:
            s.append(point[4])
        if len(point) >= 6:
            o.append(point[5]*(1/rand))

    fig = plt.figure()
    if len(point) <= 2:
        if len(point) == 1:
            y = [0] * len(x)
```

```
            plt.scatter(x, y, c='black', alpha=1)
            plt.scatter(point1[0], 0, c='blue')
            plt.scatter(point2[0], 0, c='blue')
            xLine = [point1[0], point2[0]]
            yLine = [0, 0]
            plt.plot(xLine, yLine)
        if len(point) == 2:
            plt.scatter(x, y, c='black', alpha=1)
            plt.scatter(point1[0], point1[1], c='blue')
            plt.scatter(point2[0], point2[1], c='blue')
            xLine = [point1[0], point2[0]]
            yLine = [point1[1], point2[1]]
            plt.plot(xLine, yLine)
        plt.show()
    elif len(point) <= 6:
        ax = fig.add_subplot(111, projection='3d')
        if len(point) == 3:
            x.remove(point1[0])
            x.remove(point2[0])
            y.remove(point1[1])
            y.remove(point2[1])
            z.remove(point1[2])
            z.remove(point2[2])
            ax.scatter(x, y, z, c='black', alpha=1)
            ax.scatter(point1[0], point1[1], point1[2], c='blue')
            ax.scatter(point2[0], point2[1], point2[2], c='blue')
        if len(point) == 4:
            img = ax.scatter(x, y, z, c=c, cmap=plt.hot(), alpha=1)
            fig.colorbar(img)
        if len(point) == 5:
            img = ax.scatter(x, y, z, c=c, cmap=plt.hot(), s=s, alpha=1)
            fig.colorbar(img)
        if len(point) == 6:
            img = ax.scatter(x, y, z, c=c, cmap=plt.hot(), s=s, alpha=o)
            fig.colorbar(img)

        xLine = [point1[0], point2[0]]
        yLine = [point1[1], point2[1]]
        zLine = [point1[2], point2[2]]
        ax.plot(xLine, yLine, zLine, c='blue')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set zlabel('Z')
        plt.show()
    else:
        print("\nTidak dapat divisualisasikan!")
```

3. Input dan Output

Eksperimen input dan output dilakukan pada spesifikasi komputer “AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz”. Angka yang digunakan untuk meng-*generate* titik-titik random hanya dibatasi sampai 1000.

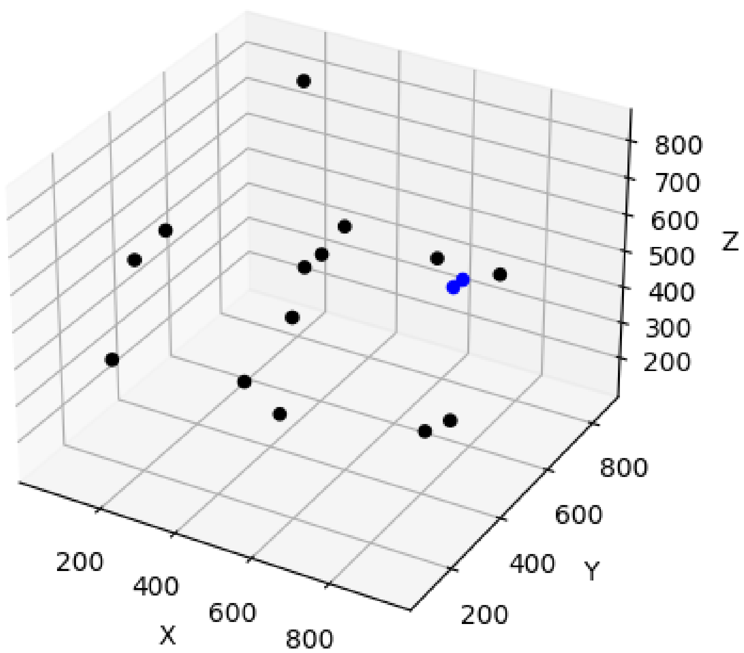
3.1 **16 titik**



```
Masukkan jumlah titik: 16
Masukkan jumlah dimensi: 3
Dua titik yang paling berdekatan:
Titik 1: 903.81, 346.22, 717.72
Titik 2: 938.79, 329.04, 757.29

Jaraknya adalah: 55.53
Banyaknya operasi perhitungan rumus Euclidian:  41
Waktu eksekusi: 0.00 ms
AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```

Jumlah operasi *Euclidian Distance* yang digunakan pada algoritma *Divide and Conquer* hanyalah 41 kali. Sedangkan jika menggunakan *Brute Force*, jumlah pengoprasian *Euclidian Distance* adalah sebanyak 120 kali. Dalam kasus ini, algoritma *Divide and Conquer* berhasil mengurangi langkah-langkah pengerjaan *Brute Force* yaitu sebanyak 66%.

Visualisasi 3D:

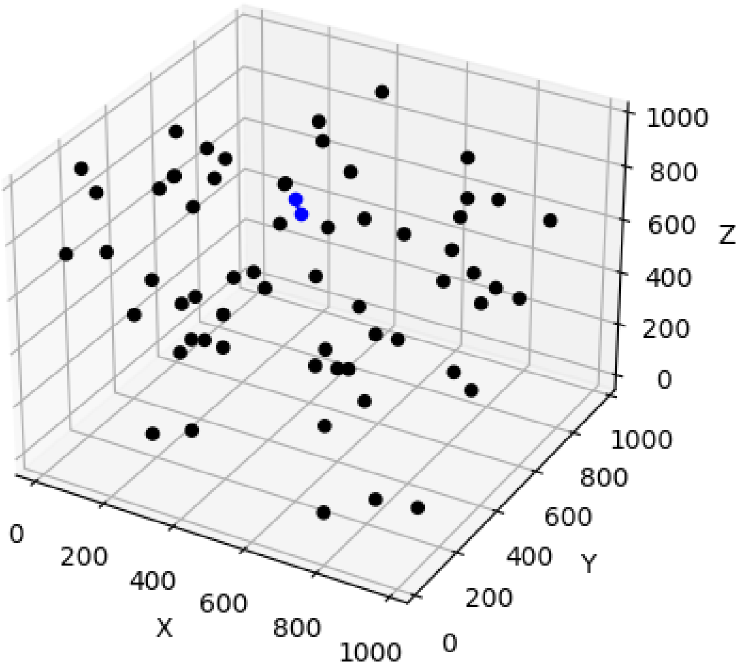


3.2 **64 titik**

```
--------------------------------------------------------------------------------
Masukkan jumlah titik: 64
Masukkan jumlah dimensi: 3
--------------------------------------------------------------------------------
Dua titik yang paling berdekatan:
Titik 1: 420.52, 496.71, 797.43
Titik 2: 458.97, 460.37, 782.05

Jaraknya adalah: 55.09
Banyaknya operasi perhitungan rumus Euclidian:  820
--------------------------------------------------------------------------------
Waktu eksekusi: 1.00 ms
AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```

Jumlah operasi *Euclidian Distance* yang digunakan pada algoritma *Divide and Conquer* hanyalah 820 kali. Sedangkan jika menggunakan *Brute Force*, jumlah pengoprasian *Euclidian Distance* adalah sebanyak 2016 kali. Dalam kasus ini, algoritma *Divide and Conquer* berhasil mengurangi langkah-langkah pengerjaan *Brute Force* yaitu sebanyak 59%.

Visualisasi 3D:

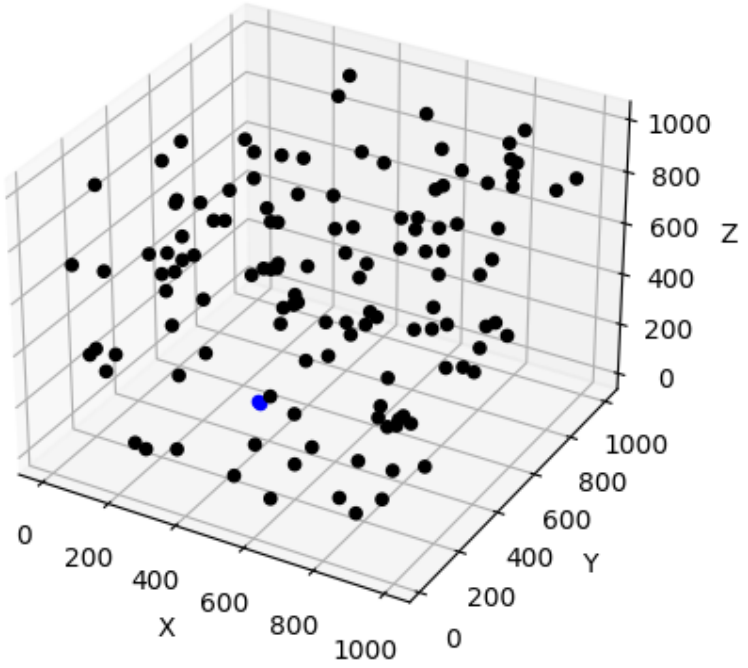


3.3 **128 titik**

```
  ___       _                                  _       ___                  _
 / __| ___ | | __ _  _ __    __ _  | |_  |  \  __ _ | |_  __ _  _ _   __ _
 \__ \/ -_)| |/ _` || '  \ / _` ||  _| | |) |/ _` ||  _|/ _` || ' \ / _` |
 |___/\___||_|\__,_||_|_|_|\__,_| \__| |___/ \__,_| \__|\__,_||_||_|\__, |
                                                                     |___/
    _ _   ___
  __| |(_) | _ \ _ _   ___  __ _  _ _   __ _  _ __
 / _` || | |  _/| '_|/ _ \/ _` || '_|/ _` || '  \
 \__,_||_| |_|  |_|  \___/\__, ||_|  \__,_||_|_|_|
                          |___/
  _ _  ___  _                    _          _     ___   _      _                                  _ _
 ( | )/ __|| |_    ___  _ _ | |_  ___  ___| |_  |   \ (_) ___| |_  __ _  _ _   __  ___( | )
  V V \__ \| ' \ / _ \| '_||  _|/ -_)(_-<|  _| | |) || |(_-<|  _|/ _` || ' \ / _|/ -_)V V
      |___/|_||_|\___/|_|   \__|\___|/__/ \__| |___/ |_|/__/ \__|\__,_||_||_|\__|\___|

------------------------------------------------------------------------------------------------------------
Masukkan jumlah titik: 128
Masukkan jumlah dimensi: 3
------------------------------------------------------------------------------------------------------------
Dua titik yang paling berdekatan:
Titik 1: 449.69, 268.34, 216.42
Titik 2: 455.00, 272.13, 205.87

Jaraknya adalah: 12.41
Banyaknya operasi perhitungan rumus Euclidian:  1698
------------------------------------------------------------------------------------------------------------
Waktu eksekusi: 2.00 ms
AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```

Jumlah operasi *Euclidian Distance* yang digunakan pada algoritma *Divide and Conquer* hanyalah 1698 kali. Sedangkan jika menggunakan *Brute Force*, jumlah pengoprasian *Euclidian Distance* adalah sebanyak 8128 kali. Dalam kasus ini, algoritma *Divide and Conquer* berhasil mengurangi langkah-langkah pengerjaan *Brute Force* yaitu sebanyak 79%.

Visualisasi 3D:

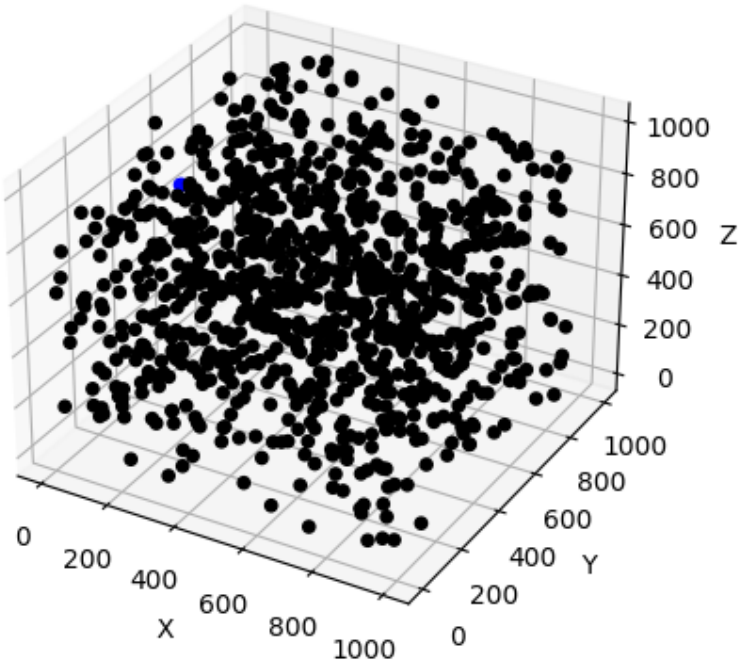


3.4 **1000 titik**

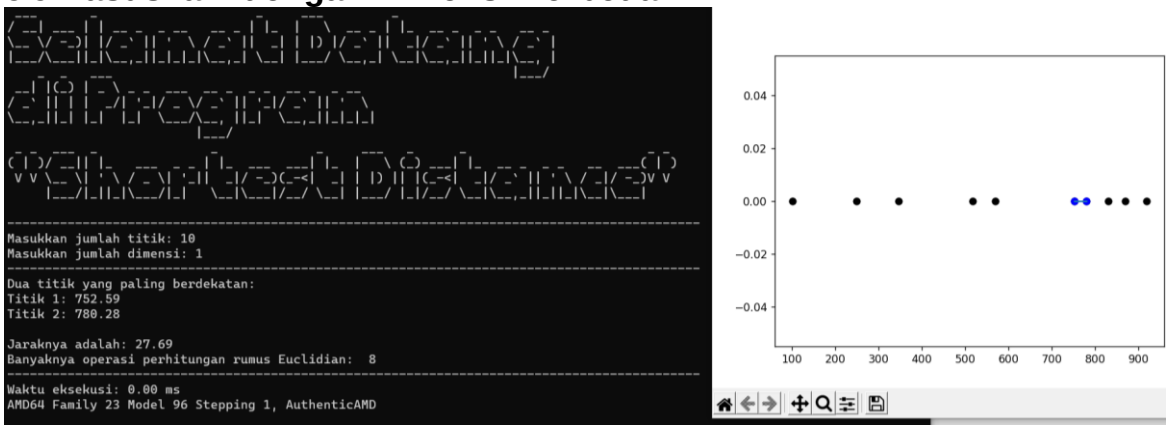
```
 ___      _                                  _     ___                  _
/ __| ___ | | __ _  _ __    __ _  | |_  |   \ __ _ | |_  __ _  _ _   __ _
\__ \/ -_)| |/ _` || '  \ / _` ||  _| | |) |/ _` ||  _|/ _` || ' \ / _` |
|___/\___||_|\__,_||_|_|_|\__,_| \__| |___/ \__,_| \__|\__,_||_||_|\__, |
                                                                    |___/
    _  _    ___                                                                 _ _
 __| |(_) | _ \ _ _  ___  __ _  _ _  __ _  _ __
/ _` || | |  _/| '_|/ _ \/ _` || '_|/ _` || '  \
\__,_||_| |_|  |_|  \___/\__, ||_|  \__,_||_|_|_|
                         |___/
 _ _  ___  _                                  _      ___   _      _                                  _ _
( | )/ __|| |_   ___  _ _  | |_  ___  ___  | |_   |   \ (_) ___| |_  __ _  _ _   __  ___  ___ ( | )
 V V \__ \| ' \ / _ \| '_||  _|/ -_)(_-<|  _| | |) || |(_-<|  _|/ _` || ' \ / _|/ -_)V V
     |___/|_||_|\___/|_|   \__|\___|/__/ \__| |___/ |_|/__/ \__|\__,_||_||_|\__|\___|

------------------------------------------------------------------------------------------------------------
Masukkan jumlah titik: 1000
Masukkan jumlah dimensi: 3
------------------------------------------------------------------------------------------------------------
Dua titik yang paling berdekatan:
Titik 1: 8.57, 600.40, 671.76
Titik 2: 12.96, 605.78, 671.99

Jaraknya adalah: 6.95
Banyaknya operasi perhitungan rumus Euclidian:  43994
------------------------------------------------------------------------------------------------------------
Waktu eksekusi: 46.05 ms
AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```

Jumlah operasi *Euclidian Distance* yang digunakan pada algoritma *Divide and Conquer* hanyalah 43994 kali. Sedangkan jika menggunakan *Brute Force*, jumlah pengoprasian *Euclidian Distance* adalah sebanyak 499500 kali. Dalam kasus ini, algoritma *Divide and Conquer* berhasil mengurangi langkah-langkah pengerjaan *Brute Force* yaitu sebanyak 91%.

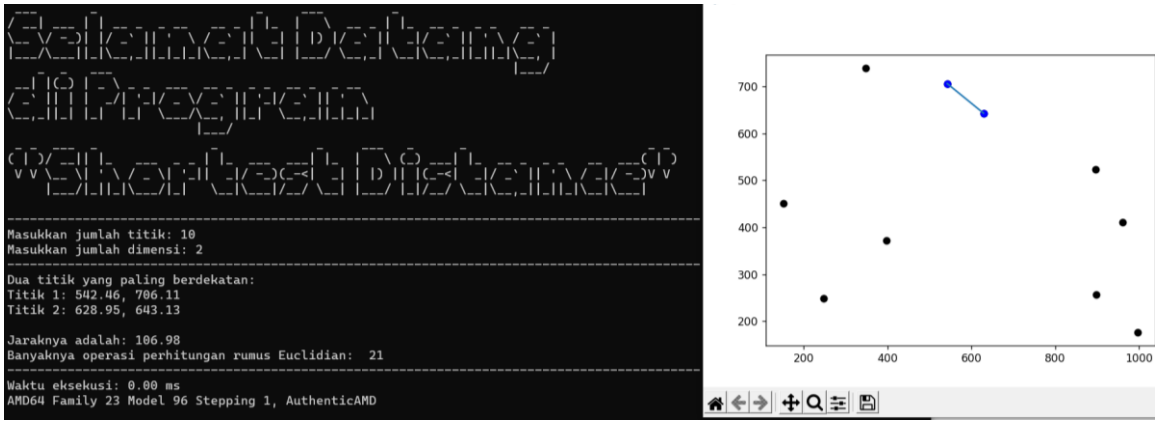
Visualisasi 3D:



3.5 **Kasus lain dengan Dimensi Berbeda**



```
Masukkan jumlah titik: 10
Masukkan jumlah dimensi: 1
----------
Dua titik yang paling berdekatan:
Titik 1: 752.59
Titik 2: 780.28

Jaraknya adalah: 27.69
Banyaknya operasi perhitungan rumus Euclidian:  8
----------
Waktu eksekusi: 0.00 ms
AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```



```
Masukkan jumlah titik: 10
Masukkan jumlah dimensi: 2
----------
Dua titik yang paling berdekatan:
Titik 1: 542.46, 706.11
Titik 2: 628.95, 643.13

Jaraknya adalah: 106.98
Banyaknya operasi perhitungan rumus Euclidian:  21
----------
Waktu eksekusi: 0.00 ms
AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```

```
Masukkan jumlah titik: 10
Masukkan jumlah dimensi: 4
----------
Dua titik yang paling berdekatan:
Titik 1: 413.27, 583.94, 280.61, 407.55
Titik 2: 518.95, 695.38, 483.72, 560.34

Jaraknya adalah: 296.97
Banyaknya operasi perhitungan rumus Euclidian:  39
----------
Waktu eksekusi: 0.00 ms
AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```

*Catatan = Value pada dimensi keempat ditandai oleh heatmap, semakin gelap warna semakin kecil valuenya, semakin terang semakin besar valuenya. Walapun representasi 4 dimensi yang sebenarnya bukan seperti gambar di atas karena tidak mungkin untuk memvisualisasikan 4 dimensi dan seterusnya.

*Catatan = Value pada dimensi kelima ditandai oleh size bola, semakin kecil volume bola semakin kecil valuenya, semakin besar volume bola semakin besar valuenya. Walapun representasi 5 dimensi yang sebenarnya bukan seperti gambar di atas karena tidak mungkin untuk memvisualisasikan 4 dimensi dan seterusnya.

*Catatan = Value pada dimensi keenam ditandai oleh transparansi bola, semakin transparan semakin kecil valuenya, semakin tidak transparan semakin besar valuenya. Walapun representasi 6 dimensi yang sebenarnya bukan seperti gambar di atas karena tidak mungkin untuk memvisualisasikan 4 dimensi dan seterusnya.

4. Link Repo

https://github.com/bernarduswillson/Tucil2_13521021.git

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa ada 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	