Tugas Kecil 2 IF2211 Strategi Algoritma

Semester II tahun 2022/2023

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

Disusun oleh:

Asyifa Nurul Shafira	13521125
Ferindya Aulia Berlianty	13521161

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

DAFTAR GAMBAR

BAB 1

DESKRIPSI TUGAS

Mencari sepasang titik terdekat dengan Algoritma Divide and Conquer sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Pada Tucil 2 kali ini Anda diminta mengembangkan algoritma mencari sepasang titik terdekat pada bidang 3D. Misalkan terdapat n buah titik pada ruang 3D. Setiap titik P di dalam ruang dinyatakan dengan koordinat P = (x, y, z). Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titik P1 = (x1, y1, z1) dan P2 = (x2, y2, z2) dihitung dengan rumus Euclidean berikut:

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

Buatlah program dalam dalam Bahasa C/C++/Java/Python/Golang/Ruby/Perl (pilih salah satu) untuk mencari sepasang titik yang jaraknya terdekat satu sama lain dengan menerapkan algoritma divide and conquer untuk penyelesaiannya, dan perbandingannya dengan Algoritma Brute Force.

Masukan program:

- n
- titik-titik (dibangkitkan secara acak) dalam koordinat (x, y, z)

Luaran program

- sepasang titik yang jaraknya terdekat dan nilai jaraknya
- banyaknya operasi perhitungan rumus Euclidean
- waktu riil dalam detik (spesifikasikan komputer yang digunakan)
- Bonus 1 (Nilai = 7,5) penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya.

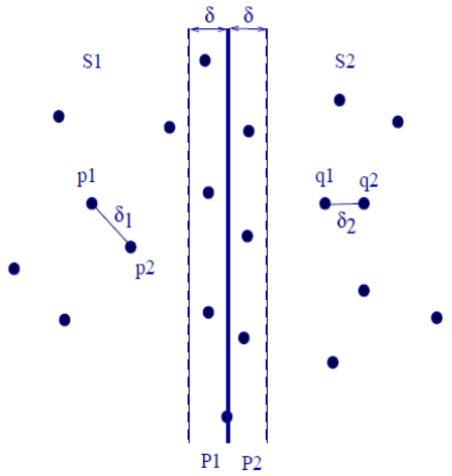
Bonus 2 (nilai = 7,5): Generalisasi program anda sehingga dapat mencari sepasang titik terdekat untuk sekumpulan vektor di Rn , setiap vektor dinyatakan dalam bentuk x = (x1, x2, …, xn)

BAB 2

LANDASAN TEORI

2.1 Penjelasan Algoritma *Divide and Conquer*

Strategi *divide and conquer* merupakan suatu metode penyelesaian masalah dengan cara membagi masalah menjadi beberapa sub masalah yang lebih kecil dan sederhana, kemudian menyelesaikan masing-masing sub masalah tersebut, lalu menggabungkan kembali menjadi algoritma utuh. Umumnya, strategi divide and conquer digunakan untuk menyelesaikan permasalahan yang dapat dibagi menjadi beberapa sub masalah serupa yang lebih kecil dari permasalahan awalnya dan independen satu sama lain, kemudian solusi-solusi yang dihasilkan dari sub permasalahan tersebut ketika digabungkan dapat membuat solusi permasalahan yang lebih besar. Strategi *divide and conquer* biasanya digunakan pada permasalahan penggunaan struktur data *collection* yang memiliki properti rekursif, seperti *dynamic list* dan *tree*. Selain struktur data rekursif, strategi *divide and conquer* juga sering digunakan pada permasalahan geometri komputasional, seperti *closest pair of points*.



Gambar 2.1 Contoh kasus closest pair dan titik-titik yang terletak di sekitar garis pembagi

Di dalam implementasinya, strategi *divide and conquer* berkaitan erat dengan implementasi algoritma rekursif. Algoritma rekursif merupakan algoritma yang terdiri atas dua bagian yaitu *base case* dan *recurrence*. *Base case* dari suatu algoritma rekurens merupakan kondisi dimana rekursivitas algoritma tersebut berhenti. Dalam banyak kasus, *base case* dari algoritma rekursif merupakan *initial value* atau kasus terkecil yang tidak dapat dibagi lebih lanjut. *Recurrence* dari algoritma rekurens merupakan kondisi dimana terdapat pemanggilan kembali algoritma rekurens tersebut, pemanggilan dalam kasus ini dapat

diartikan sebagai pemanggilan fungsi, prosedur, *method*, ataupun ADT rekursif dengan nilai masukan yang mendekati b*ase case*. Jika nilai masukan tidak berubah atau bahkan menjauhi *base case*, maka akan terjadi infinite recursion yang akan menyebabkan permasalahan tidak dapat diselesaikan.

Strategi penyelesaian dengan algoritma divide and conquer tidak sesederhana algoritma brute force. Penyelesaian dengan divide and conquer membutuhkan pencapaian sebuah lemma yang bergantung pada strategi penyelesaian itu sendiri. Lemma inilah yang menjadi kunci optimasi. Pada konsepnya, penyelesaian menggunakan algoritma divide and conquer dapat diabstraksikan melalui pseudocode berikut:

```
function findClosestPair(left , right) : real
if (right = left)
   return 0
else
   l = findClosestPair(left , (left + right) / 2)
   r = findClosestPair((left + right) / 2 + 1 , right)
   mid = conquer(left , right)
   return min(l,r,mid)
```

Gambar 2.2 Pseudocode closest pair of points dengan algoritma divide and conquer

Dalam implementasinya strategi divide and conquer dapat dibagi menjadi 3 bagian, yaitu divide, conquer, serta merge. Semua langkah tersebut akan dijelaskan lebih jelas di bawah ini:

1. Divide

   Strategi ini membagi permasalahan menjadi beberapa sub permasalahan yang identik, berukuran hampir sama dengan satu sama lain, independen dengan sub persoalan lainnya, serta pembagian partisi sub permasalahan konsisten. Pada pembagian permasalahan sederhana seperti algoritma *insertion sort* maupun *merge sort*, algoritma partisi yang dilakukan pada masukan tidak terlalu kompleks. Algoritma partisi tersebut dinamakan sebagai *easy split/hard join*. Tetapi, untuk permasalahan pembagian lebih kompleks, seperti *selection sort*, *quicksort*, *quickhull*, maupun *closest pair of points*, algoritma partisi yang dilakukan untuk memecah permasalahan akan menjadi jauh lebih kompleks. Pada permasalahan tersebut, dibutuhkan algoritma yang telah terbukti paling efektif secara matematis untuk membagi permasalahan tersebut (sebagai contoh penggunaan algoritma partisi Hoare untuk menyelesaikan permasalahan *quick sort*).

2. Conquer

Strategi ini menyelesaikan masing-masing sub persoalan dengan langsung melakukan perhitungan solusi jika sub persoalan tersebut termasuk *base case* dari algoritma rekursif atau membaginya kembali menjadi sub persoalan - persoalan baru yang diselesaikan secara rekursif apabila sub persoalan masih terlalu besar atau termasuk sebagai recurrence dari algoritma rekursif. Pembuat program dapat menambahkan beberapa manipulasi algoritma untuk mengefisienkan rekursivitas algoritma. Biasanya, pemrogram dapat melakukan memoisasi atau pencatatan hasil dari hasil algoritma rekursif yang telah dilakukan terlebih dahulu oleh pemrogram. Contoh dari algoritma yang menggunakan strategi conquer tersebut yaitu algoritma *binary exponentiation* dimana pembuat program hanya melakukan satu kali *recurrence* untuk dua sub persoalan yang berbeda. Selain itu, pemrogram juga dapat mengurangi jumlah sub persoalan yang akan diteruskan ke dalam *recurrence* dengan melakukan manipulasi aljabar. Contoh dari algoritma yang menggunakan strategi conquer tersebut yaitu *Strassen's matrix multiplication* dan *Karatsuba algorithm*.

3. Merge

Strategi ini menggabungkan seluruh solusi yang didapatkan dari algoritma rekursif dengan masukan sub persoalan sehingga dapat membentuk solusi untuk menyelesaikan permasalahan. Seperti pada tahap divide, terdapat beberapa algoritma penggabungan yang tingkat kompleksitasnya berbeda-beda tergantung dari jenis algoritmanya itu sendiri. Terdapat pola menarik yang terlihat pada tahap ini, ketika suatu permasalahan memiliki strategi divide and conquer dengan algoritma partisi yang cukup sederhana, maka akan dipastikan bahwa algoritma penggabungan yang dimiliki oleh strategi tersebut akan cukup kompleks. Hal ini terbukti untuk algoritma-algoritma sederhana seperti merge sort dan insertion sort dimana algoritma penggabungan solusi dari sub persoalannya tidak sederhana. Sebaliknya, apabila suatu permasalahan memiliki strategi divide and conquer dengan algoritma partisi yang kompleks maka dipastikan bahwa algoritma penggabungan yang dimiliki oleh strategi tersebut akan cukup mudah. Hal ini terbukti untuk algoritma-algoritma yang cukup kompleks, seperti selection sort, quick sort, quickhull, dan closest pair of points.

Seperti yang telah dijelaskan sebelumnya, strategi divide and conquer mengimplementasikan algoritma rekursif untuk menyelesaikan permasalahannya. Algoritma rekursif akan memanggil dirinya sendiri sebagai solusi dari masukan yang diberikan. Oleh

karena itu, muncul suatu permasalahan baru ketika penulis ingin melakukan analisis terhadap strategi divide and conquer. Hal ini disebabkan oleh rekursivitas algoritmanya, maka untuk kompleksitas waktu dari algoritma tersebut juga memiliki bentuk rekursif. Hal tersebut menyulitkan penulis untuk memformulasikan notasi Big O untuk menentukan seberapa efektif strategi divide and conquer yang telah dibuat. Oleh karena itu, penulis menggunakan Teorema Master untuk membantu penulis mencari kompleksitas waktu dari algoritma rekursif. Berikut ini merupakan penjelasan mengenai Teorema Master. Misalkan T(n) adalah fungsi monoton naik yang memenuhi relasi rekurens:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

Dengan asumsi bahwa $n = b^k$, $k \in \{1, 2, 3, \dots\}$ serta $a \geq 1,\ b \geq 2,\ c \geq 0,\ dan\ d \geq 0$
Maka, notasi Big O dari fungsi monoton $T(n)$ adalah:

$$T(n) = \begin{cases} O(n^d), & a < b^d \\ O(n^d \cdot \log n), & a = b^d \\ O\left(n^{\log_b a}\right), & a < b^d \end{cases}$$

Permasalahan yang diangkat pada tugas kecil ini adalah memvisualisasikan titik yang terdapat pada sebuah bidang 3D dan menganalisis jarak pasangan titik yang jaraknya terdekat satu sama lain dengan menerapkan algoritma divide and conquer untuk penyelesaiannya, dan perbandingannya dengan Algoritma Brute Force.

Closest pair of points merupakan pasangan titik yang terdekat atau pasangan titik yang mempunyai jarak minimal di antara pasangan titik yang lain pada bidang dengan ruang dimensi tertentu. Dengan kata lain, jika terdapat n buah titik pada bidang dengan dimensi tertentu, maka *closest pair of points* merupakan pasangan titik dengan jarak terdekat satu sama lain dibandingkan dengan pasangan titik lainnya. Permasalahan closest pair merupakan salah satu permasalahan klasik dalam dunia matematika diskrit. Permasalahan ini sering dikaitkan dengan permasalahan pada dunia sehari-hari, contohnya seperti proses pengambilan dua bahan baku oleh lengan robot pada pabrik mikroprosesor agar pergerakan lengan robot seminimal mungkin. Secara naif, permasalahan closest pair dapat diselesaikan dengan algoritma brute force yang membutuhkan kompleksitas waktu $O\ (n^2)$. Untuk n yang besar (n > 100000), komputasi dengan algoritma brute force akan memakan waktu yang lama, bahkan dengan komputer paling cepat saat ini.

Oleh karena itu, penulis dalam tugas kecil ini melakukan perbandingan algoritma brute force dengan algoritma divide and conquer yang memiliki kompleksitas $O\ (n\ log\ n)$. Untuk n yang besar, algoritma ini masih dirasa cukup mangkus untuk diterapkan dalam penyelesaian masalah. Meskipun demikian, pencapaian algoritma divide and conquer membutuhkan beberapa lemma yang terbatas pada parameter tertentu.

2.2 Penjelasan Algoritma Brute Force

Algoritma brute force merupakan penyelesaian suatu masalah dengan pendekatan yang sederhana, langsung, jelas, dan mudah dipahami. Dengan menggunakan algoritma brute force, penyelesaian suatu masalah cenderung dilakukan dengan mencari dan meninjau semua kasus yang ada. Dengan demikian, algoritma brute force dapat dikatakan sebagai algoritma yang lempang (*straightforward*). Keunggulan dari algoritma ini adalah cukup mudah untuk dipahami dan dapat diterapkan untuk hamper semua permasalahan komputasi. Akan tetapi, kekurangan dari algoritma ini adalah tidak efisien karena membutuhkan langkah yang banyak dalam penyelesaiannya sehingga membutuhkan waktu yang cukup lama.

Solusi brute force untuk masalah ini cukup straightforward yaitu dengan cara tinjau seluruh kemungkinan pasangan titik-titik kemudian hitung jaraknya dan cari nilai maksimum. Berikut pseudocode untuk solusi brute force:

```
function findClosestPair : real
min = ~
foreach i in pointList
 foreach j in pointList
 if (i <> j AND distance(i , j) < max)
 min = distance(i , j)
return min
```

Gambar 2.3 Pseudocode closest pair of points dengan algoritma brute force

Jadi, untuk setiap instance permasalahan akan terjadi n pass untuk setiap titik dan masing-masing pass akan melakukan peninjauan kembali untuk n titik. Sehingga kompleksitas waktu untuk algoritma ini adalah $O\ (n^2)$.

Untuk n sama dengan 100000 dengan koordinat titik acak yang berada pada rentang -100000 hingga 100000, penulis memperoleh hasil program berjalan dalam waktu yang lebih sebentar. Jika algoritma ini diterapkan dalam dunia sehari-hari, algoritma ini tetap dapat memberikan jawaban yang benar tetapi tidak dapat berjalan secara mangkus dan sangkil.

BAB 3

IMPLEMENTASI

3.1 Algoritma Divide and Conquer

Algoritma divide and conquer yang kami gunakan untuk mencari dua titik dengan jarak terdekat pada dimensi 3 adalah sebagai berikut:

1. Sorting titik berdasarkan koordinat x dan y.
2. Jika jumlah titik kurang dari atau sama dengan 3 maka program akan menggunakan metode brute force untuk mencari pasangan titik dengan jarak terdekat dari setiap kemungkinan pasangan titik yang ada.
3. Jika jumlah titik lebih dari 3, maka program akan menggunakan metode divide and conquer untuk menyelesaikannya.
4. Pertama, titik akan dibagi relatif merata ke dalam dua bagian ruang.
5. Kemudian cari pasangan titik terdekat pada setiap bagian ruang secara rekursif. Jarak pasangan titik terdekat pada bagian ruang kiri akan disimpan pada leftDistance dan jarak pasangan titik terdekat pada bagian ruang kanan akan disimpan pada rightDistance.
6. Program akan membandingkan jarak antara leftDistance dan rightDistance untuk menjadi pasangan titik terdekat.
7. Selanjutnya, program menghitung jarak pasangan titik yang melewati garis tengah (lintas bagian ruang) menggunakan euclidean distance.
8. Terakhir, dilakukan perbandingan antara hasil pada poin 7 dengan hasil pada poin 6 dan jarak terdekat akan ditampilkan.

3.2 Source Code dengan Bahasa Python

a. main.py

```python
import math
import random
from time import time
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

# Algoritma Brute Force untuk mencari pasangan titik terdekat
def findDistanceBruteForce(points):
    n = len(points)
    minDistance = float('inf')
    closestPair = None

    for i in range(n):
        for j in range(i+1, n):
            d = findDistance(points[i], points[j])
            if d < minDistance:
                minDistance = d
                closestPair = (points[i], points[j])

    return closestPair, minDistance

# Algoritma Divide and Conquer untuk mencari pasangan titik terdekat
def findDistanceDivideConquer(points):
    n = len(points)
    if n <= 3:
        return findDistanceBruteForce(points)

    # Sort points berdasarkan koordinat X
    points = sorted(points, key=lambda x: x[0])

    # Divide points menjadi 2 bagian
    mid = n // 2
    left = points[:mid]
    right = points[mid:]
```

```
    # Mencari pasangan terdekat di setiap bagian secara rekursif
    leftPair, leftDistance = findDistanceDivideConquer(left)
    rightPair, rightDistance = findDistanceDivideConquer(right)

    # Mencari pasangan titik terdekat di kedua bagian
    if leftDistance < rightDistance:
        closestPair = leftPair
        minDistance = leftDistance
    else:
        closestPair = rightPair
        minDistance = rightDistance

    # Mencari pasangan titik terdekat yang melintasi garis tengah
    strip = []
    for point in points:
        if abs(point[0] - points[mid][0]) < minDistance:
            strip.append(point)
    strip = sorted(strip, key=lambda x: x[1])

    for i in range(len(strip)):
        for j in range(i+1, len(strip)):
            if strip[j][1] - strip[i][1] >= minDistance:
                break
            d = findDistance(strip[i], strip[j])
            if d < minDistance:
                minDistance = d
                closestPair = (strip[i], strip[j])

    return closestPair, minDistance

# Fungsi untuk menghitung jarak antara 2 titik dalam ruang 3D
def findDistance(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2 + (p1[2] - p2[2])**2)
```

```
# ASCII ART
print("      \:.            .:/                                                ")
print("       \``._________.''/                                                ")
print("        \            /                                                 ")
print(" .---.--, / .':.   .':. \                                              ")
print("/__:  /  | '::' . '::' |       === WELCOME TO CLOSEST PAIR OF POINTS PROGRAM ===")
print("   / /   |`.   ._.   .'|                    BY 13521161 & 13521125      ")
print("  / /    |.'         '.|                                                ")
print(" /___-_-,|.\  \   /  /.|                                                ")
print("     // |''\.;   ;,/ '|                                                ")
print("     `==|:=         =:|                                                ")
print("        `.           .'                                                ")
print("          :-._____.-:                                                  ")
print("         ```       ```                                                 ")
print("                                                                       ")

# Generate secara random 100 titik dalam ruang 3D
num_points = int(input("Masukkan jumlah titik pada ruang 3D (n) : "))
points = [(random.uniform(-100, 100), random.uniform(-100, 100), random.uniform(-100, 100)) for i in range(num_points)]

# Mencari pasangan titik terdekat menggunakan algoritma Brute Force
startTime = time()
bruteForcePair, bruteForceDistance = findDistanceBruteForce(points)
bruteForceTime = (time() - startTime)
bruteForceOp = num_points**2/2

# Mencari pasangan titik terdekat menggunakan algoritma Divide and Conquer
startTime = time()
divideConquerPair, divideConquerDistance = findDistanceDivideConquer(points)
divideConquerTime = (time() - startTime)
divideConquerOp = num_points * math.log(num_points, 2)
```

```
# Mencetak pasangan titik terdekat beserta nilai jarak untuk algoritma brute force dan divide and conquer
print("-----------------------------------------------------------------[[ALGORITMA BRUTE FORCE]]-----------------------------------------------------------------")
print("Pasangan titik terdekat : ", bruteForcePair)
print("Nilai jarak : ", bruteForceDistance)
print("Banyaknya operasi perhitungan rumus Euclidean : ", bruteForceOp)
print("Waktu Eksekusi : ", bruteForceTime, "detik")
print("                                                                                                                                                    ")
print("-----------------------------------------------------------[[ALGORITMA DIVIDE AND CONQUER]]-----------------------------------------------------------")
print("Pasangan titik terdekat : ", divideConquerPair)
print("Nilai jarak :", divideConquerDistance)
print("Banyaknya operasi perhitungan rumus Euclidean : ", divideConquerOp)
print("Waktu Eksekusi : ", divideConquerTime, "detik")

# Visualisasi
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter([p[0] for p in points], [p[1] for p in points], [p[2] for p in points], s=10)
ax.scatter([divideConquerPair[0][0], divideConquerPair[1][0]], [divideConquerPair[0][1], divideConquerPair[1][1]], [divideConquerPair[0][2], divideConquerPair[1][2]], s=50, c='r')
ax.set_xlabel('Sumbu X')
ax.set_ylabel('Sumbu Y')
ax.set_zlabel('Sumbu Z')
plt.show()
```

BAB 4

PENGUJIAN

4.1 Test case untuk n = 16

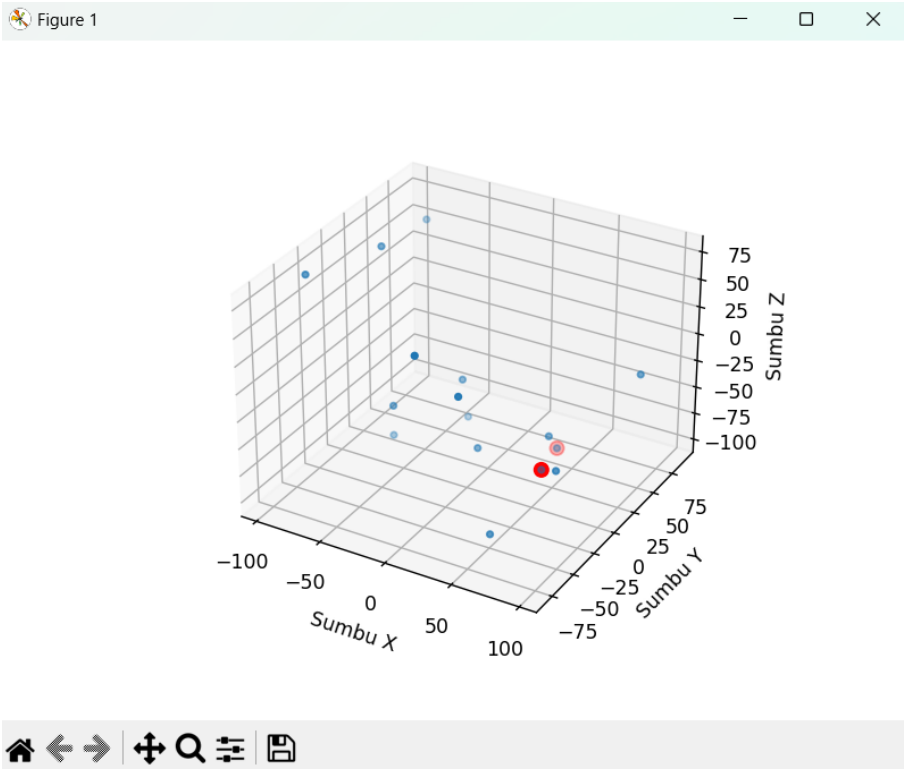
- Hasil pengujian pertama

```
PS D:\Tucil2_13521161_13521125> python -u "d:\Tucil2_13521161_13521125\src\main.py"
      \:.           .:/
       \``._______.''/
        \           /
 .---.--, / .':.   .':. \
/__:  /  | '::' . '::' |        === WELCOME TO CLOSEST PAIR OF POINTS PROGRAM ===
   / /   |`.   ._.   .'|                    BY 13521161 & 13521125
  / /    |.'        '.|
 /___-_,.|.\  \   /  /.|
     /// |''\.;   ;,/ '|
     `==|:=         =:|
        `.           .'
          :-._____.-:
          `'' `---'`''

Masukkan jumlah titik pada ruang 3D (n) : 16
------------------------------------------------------------[[ALGORITMA BRUTE FORCE]]------------------------------------------------------------
Pasangan titik terdekat :  ((41.415313873592794, 25.481794052818202, -99.0197972134220 2), (36.62615780545403, 52.60686632652329, -99.82010172556556))
Nilai jarak :  27.556234303889788
Banyaknya operasi perhitungan rumus Euclidean :  128.0
Waktu Eksekusi :  0.0 detik

------------------------------------------------------------[[ALGORITMA DIVIDE AND CONQUER]]------------------------------------------------------------
Pasangan titik terdekat :  ((41.415313873592794, 25.481794052818202, -99.0197972134220 2), (36.62615780545403, 52.60686632652329, -99.82010172556556))
Nilai jarak : 27.556234303889788
Banyaknya operasi perhitungan rumus Euclidean :  64.0
Waktu Eksekusi :  0.0 detik
```

Gambar 4.1.1 Output pertama untuk masukan n = 16



Gambar 4.1.2 Hasil visualisasi bidang 3D pertama untuk jumlah titik 16 buah

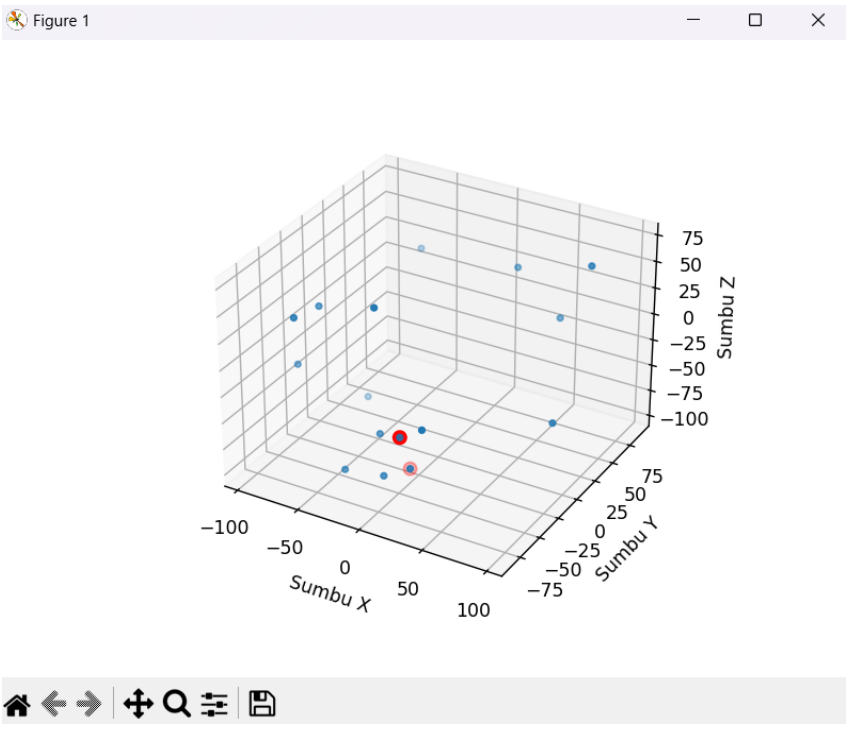
- Hasil pengujian kedua

```
PS D:\Tucil2_13521161_13521125> python -u "d:\Tucil2_13521161_13521125\src\main.py"
       \:.           .:/
        \``._________.''/
         \             /
 .---.--, /  .':.   .':. \
/__:  /   | '::' . '::' |        === WELCOME TO CLOSEST PAIR OF POINTS PROGRAM ===
   / /    |`:   ._.   :'|                      BY 13521161 & 13521125
  / /     |.'         '.|
 /___-_-,|.\  \   /  /.|
     // /|''\.;   ;./''|
      `==|:=         =:|
         `.           .'
           `:-._____.-:'
            `:,'   `:,'


Masukkan jumlah titik pada ruang 3D (n) : 16
------------------------------------------------------------[[ALGORITMA BRUTE FORCE]]------------------------------------------------------------
Pasangan titik terdekat :  ((24.790452154056823, -73.55499988716427, -60.44162858336992), (14.565321690699975, -70.28142947686442, -36.87015228691335))
Nilai jarak :  25.90142951688715
Banyaknya operasi perhitungan rumus Euclidean :  128.0
Waktu Eksekusi :  0.0 detik

-----------------------------------------------------------[[ALGORITMA DIVIDE AND CONQUER]]-----------------------------------------------------------
Pasangan titik terdekat :  ((24.790452154056823, -73.55499988716427, -60.44162858336992), (14.565321690699975, -70.28142947686442, -36.87015228691335))
Nilai jarak : 25.90142951688715
Banyaknya operasi perhitungan rumus Euclidean :  64.0
Waktu Eksekusi :  0.0 detik
```

Gambar 4.1.3 Output kedua untuk masukan n = 16



Gambar 4.1.4 Hasil visualisasi bidang 3D kedua untuk jumlah titik 16 buah

4.2 Test case untuk n = 64

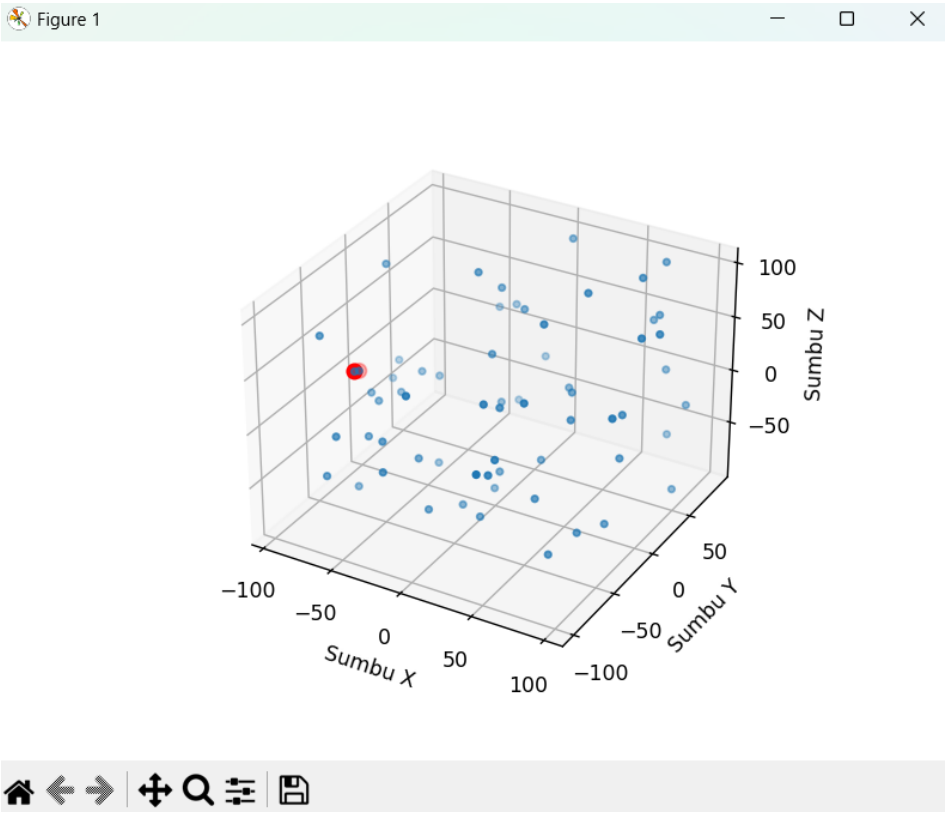
- Hasil pengujian pertama

```
PS D:\Tucil2_13521161_13521125> python -u "d:\Tucil2_13521161_13521125\src\main.py"
       \:.           .:/
        \`-._______.-'/
         \           /
 .--.--, / .':.   .':. \
/__:  /  | '::' . '::' |       === WELCOME TO CLOSEST PAIR OF POINTS PROGRAM ===
   / /   |`:   ._.   :'|                    BY 13521161 & 13521125
  / /    |.'         `.|
 /___-_-,|.\ \   /  /.|
     //  |''\.;   ;,/ '|
     `==|:=         =:|
        `.           .'
          `:-._____.-:'
           `''     ``

Masukkan jumlah titik pada ruang 3D (n) : 64
------------------------------------------------------------[[ALGORITMA BRUTE FORCE]]------------------------------------------------------------
Pasangan titik terdekat :  ((-76.03756063964464, -37.172223492490474, 21.41130505154973 2), (-75.87157709226918, -42.60566575624538, 24.42820358827467))
Nilai jarak :  6.217034836033207
Banyaknya operasi perhitungan rumus Euclidean :  2048.0
Waktu Eksekusi :  0.0010755062103271484 detik

------------------------------------------------------------[[ALGORITMA DIVIDE AND CONQUER]]------------------------------------------------------------
Pasangan titik terdekat :  ((-75.87157709226918, -42.60566575624538, 24.42820358827467), (-76.03756063964464, -37.172223492490474, 21.411305051549732))
Nilai jarak : 6.217034836033207
Banyaknya operasi perhitungan rumus Euclidean :  384.0
Waktu Eksekusi :  0.0 detik
```

Gambar 4.2.1 Output pertama untuk masukan n = 64



Gambar 4.2.2 Hasil visualisasi bidang 3D pertama untuk jumlah titik 64 buah

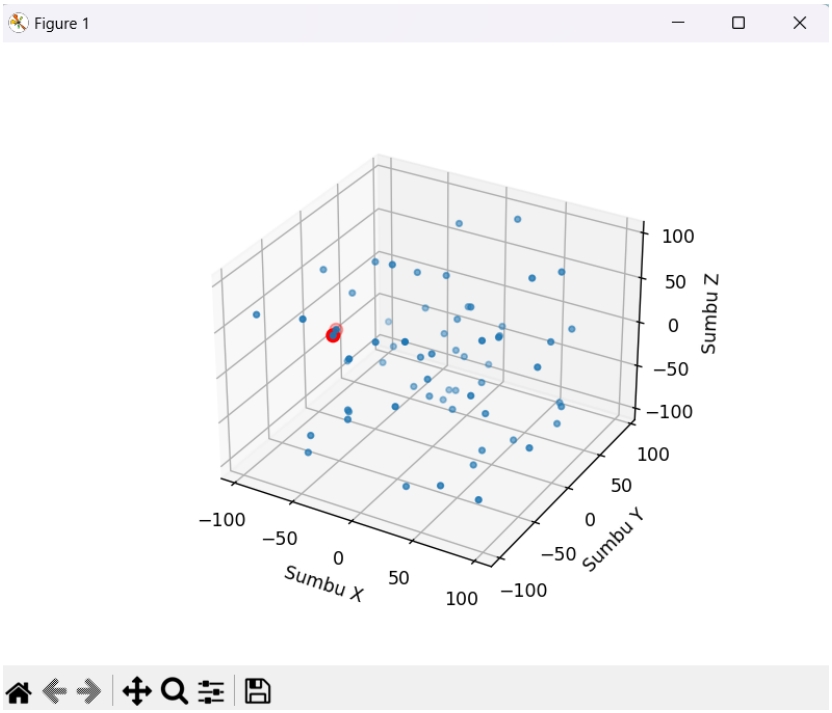
- Hasil pengujian kedua

```
PS D:\Tucil2_13521161_13521125> python -u "d:\Tucil2_13521161_13521125\src\main.py"
/__:  /   | '::' . '::' |        === WELCOME TO CLOSEST PAIR OF POINTS PROGRAM ===
   / /    |`.    ._.    .'|                    BY 13521161 & 13521125
  / /     |.'           '.|
 /___-_-,|.\  \    /  /.|
      /// |''\.;    ;,/ '|
        `==|:=         =:|
           `.           .'
             :-._____.-:
            `'''-----`''


Masukkan jumlah titik pada ruang 3D (n) : 64
------------------------------------------------------------[[ALGORITMA BRUTE FORCE]]------------------------------------------------------------
Pasangan titik terdekat :  ((-24.34246982299959, -89.41377149018086, 71.82804939218644), (-21.422210153240954, -97.8986242245469, 72.59633249718982))
Nilai jarak :  9.006159080991804
Banyaknya operasi perhitungan rumus Euclidean :  2048.0
Waktu Eksekusi :  0.0032248497009277344 detik

------------------------------------------------------------[[ALGORITMA DIVIDE AND CONQUER]]------------------------------------------------------------
Pasangan titik terdekat :  ((-21.422210153240954, -97.8986242245469, 72.59633249718982), (-24.34246982299959, -89.41377149018086, 71.82804939218644))
Nilai jarak : 9.006159080991804
Banyaknya operasi perhitungan rumus Euclidean :  384.0
Waktu Eksekusi :  0.0 detik
```

Gambar 4.2.3 Output kedua untuk masukan n = 64



Gambar 4.2.4 Hasil visualisasi bidang 3D kedua untuk jumlah titik 64 buah

4.3 Test case untuk n = 128

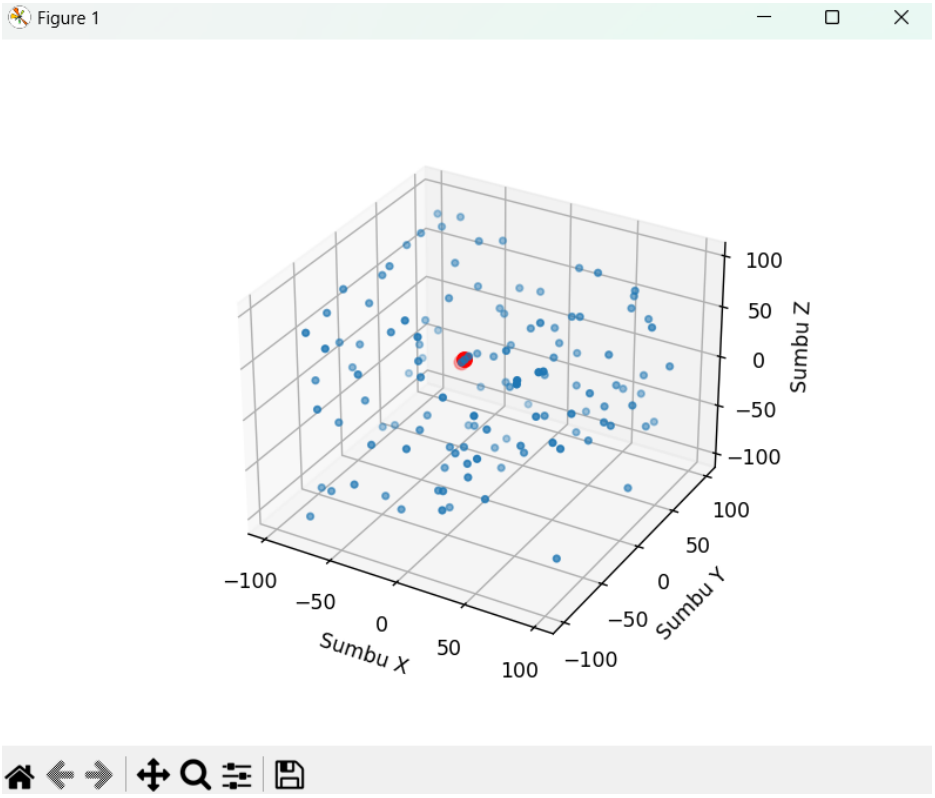
- Hasil pengujian pertama

```
PS D:\Tucil2_13521161_13521125> python -u "d:\Tucil2_13521161_13521125\src\main.py"
       \:.             .:/
        \``.________.''/
         \           /
 .--.--, / .':.   .':. \
/__:  /  | '::' . '::' |        === WELCOME TO CLOSEST PAIR OF POINTS PROGRAM ===
   / /   |`.    `_`    .'|                    BY 13521161 & 13521125
  / /    |.'          '.|
 /___-_,|.\  \    /  /.|
     // |''\.;   ;,/ '|
     `==|:=         =:|
         `.         .'
           `:-._____.-:'
            `''       ``

Masukkan jumlah titik pada ruang 3D (n) : 128
------------------------------------------------------------------[[ALGORITMA BRUTE FORCE]]------------------------------------------------------------------
Pasangan titik terdekat :  ((7.7492013159843935, -40.36049802338502, 57.11315819053803), (5.319713858122938, -40.05890746345007, 53.42356663524305))
Nilai jarak :  4.427917345508161
Banyaknya operasi perhitungan rumus Euclidean :  8192.0
Waktu Eksekusi :  0.005288124084472656 detik

---------------------------------------------------------------[[ALGORITMA DIVIDE AND CONQUER]]---------------------------------------------------------------
Pasangan titik terdekat :  ((7.7492013159843935, -40.36049802338502, 57.11315819053803), (5.319713858122938, -40.05890746345007, 53.42356663524305))
Nilai jarak : 4.427917345508161
Banyaknya operasi perhitungan rumus Euclidean :  896.0
Waktu Eksekusi :  0.0 detik
```

Gambar 4.3.1 Output pertama untuk masukan n = 128



Gambar 4.3.2 Hasil visualisasi untuk bidang 3D pertama untuk jumlah titik 128 buah

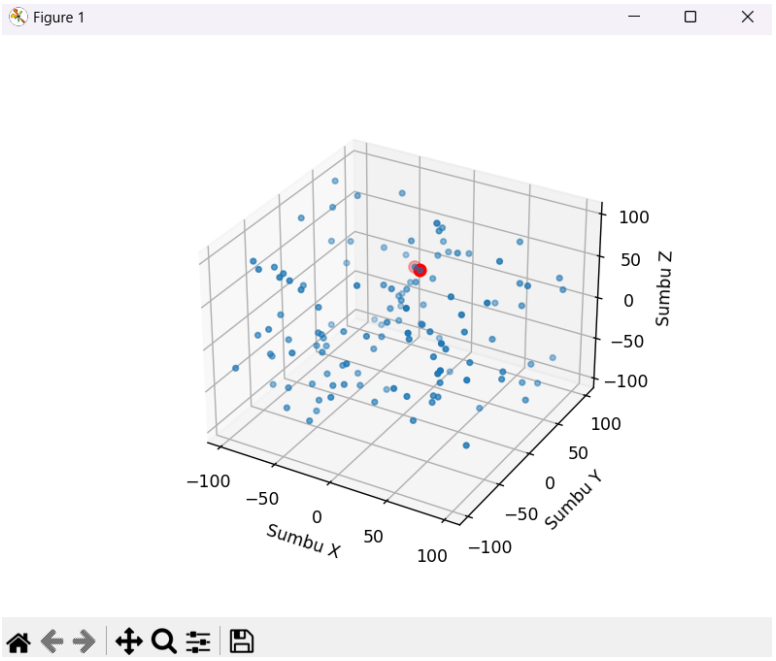
- Hasil pengujian kedua

```
PS D:\Tucil2_13521161_13521125> python -u "d:\Tucil2_13521161_13521125\src\main.py"
      \:.           .:/
       \``._______.''/
        \           /
 .--.--, / .':.   .':. \
/__:  /  | '::' . '::' |        === WELCOME TO CLOSEST PAIR OF POINTS PROGRAM ===
   / /   |`.   ._.   .'|                    BY 13521161 & 13521125
  / /    |.'         '.|
 /___-_,|.\  \   /  /.|
     //  |''\.;   ;,/ '|
     `==|:=         =:|
         `             '
          .           .
           :-._____.-:
           `''     ``'

Masukkan jumlah titik pada ruang 3D (n) : 128
------------------------------------------------------------[[ALGORITMA BRUTE FORCE]]------------------------------------------------------------
Pasangan titik terdekat :  ((-8.334777421178117, 41.348965806644856, 34.45954430775063), (-14.377126372612764, 43.62973035169702, 34.84958824915603))
Nilai jarak :  6.47023972021883
Banyaknya operasi perhitungan rumus Euclidean :  8192.0
Waktu Eksekusi :  0.004395961761474609 detik

------------------------------------------------------------[[ALGORITMA DIVIDE AND CONQUER]]------------------------------------------------------------
Pasangan titik terdekat :  ((-8.334777421178117, 41.348965806644856, 34.45954430775063), (-14.377126372612764, 43.62973035169702, 34.84958824915603))
Nilai jarak : 6.47023972021883
Banyaknya operasi perhitungan rumus Euclidean :  896.0
Waktu Eksekusi :  0.0 detik
```

Gambar 4.3.3 Output kedua untuk masukan n = 128



Gambar 4.3.4 Hasil visualisasi untuk bidang 3D kedua untuk jumlah titik 128 buah

4.4 Test case untuk n = 1000

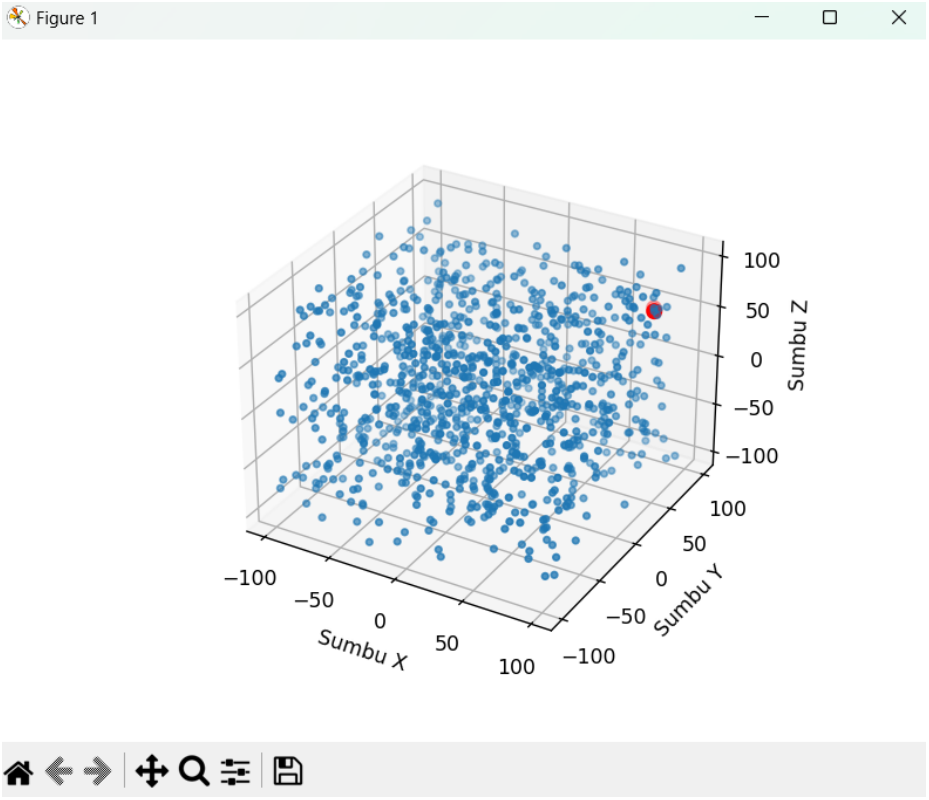
- Hasil pengujian pertama

```
PS D:\Tucil2_13521161_13521125> python -u "d:\Tucil2_13521161_13521125\src\main.py"
       \:.             .:/
        \``._________.''/
         \             /
 .--.--, / .':.   .':. \
/__:  /  | '::' . '::' |       === WELCOME TO CLOSEST PAIR OF POINTS PROGRAM ===
   / /   |`.   ._.   .'|                    BY 13521161 & 13521125
  / /    |.'         '.|
 /___-_,|.\ \   /  /.|
     // |''\.;   ;,/ '|
     `==|:=         =:|
          `.       .'
            `:-._____.-:'
             `''       ``

Masukkan jumlah titik pada ruang 3D (n) : 1000
------------------------------------------------------------[[ALGORITMA BRUTE FORCE]]------------------------------------------------------------
Pasangan titik terdekat :  ((88.106253987569, 68.8553035545055, 67.29049646303079), (89.21629896550684, 67.22595863747415, 65.59495824502665))
Nilai jarak :  2.600348930511742
Banyaknya operasi perhitungan rumus Euclidean :  500000.0
Waktu Eksekusi :  0.17948603630065918 detik

------------------------------------------------------------[[ALGORITMA DIVIDE AND CONQUER]]------------------------------------------------------------
Pasangan titik terdekat :  ((89.21629896550684, 67.22595863747415, 65.59495824502665), (88.106253987569, 68.8553035545055, 67.29049646303079))
Nilai jarak : 2.600348930511742
Banyaknya operasi perhitungan rumus Euclidean :  9965.784284662088
Waktu Eksekusi :  0.01677703857421875 detik
```

Gambar 4.4.1 Output pertama untuk masukan n = 1000



Gambar 4.4.2 Hasil visualisasi untuk bidang 3D pertama untuk jumlah titik 1000 buah

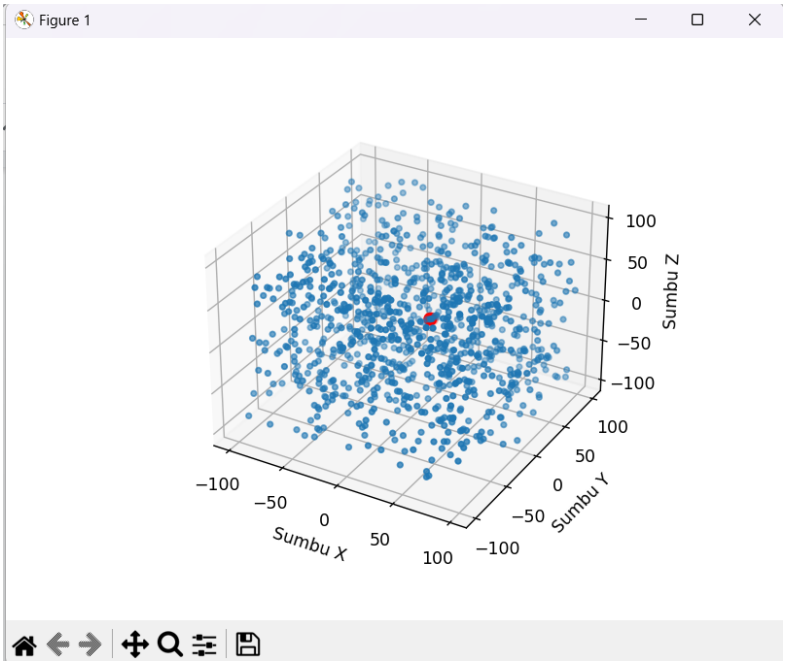
- Hasil pengujian kedua

```
PS D:\Tucil2_13521161_13521125> python -u "d:\Tucil2_13521161_13521125\src\main.py"
       \:.           .:/
        \``._______.''/
         \           /
 .--.--, / .':.   .':. \
/__:  /  | '::' . '::' |        === WELCOME TO CLOSEST PAIR OF POINTS PROGRAM ===
   / /   |`.   `-'   .'|                    BY 13521161 & 13521125
  / /    |.'         `.|
 /___-_,|.\  \   /  /.|
     // |''\.;   ;,/ '|
     `==|:=         =:|
        `.           .'
          :-._____.-:
          `'       `'

Masukkan jumlah titik pada ruang 3D (n) : 1000
------------------------------------------------------------------[[ALGORITMA BRUTE FORCE]]------------------------------------------------------------------
Pasangan titik terdekat :  ((45.13593367305393, -46.293717583225266, 56.71980347244153), (44.26009203802957, -47.44537870070915, 56.89074033502453))
Nilai jarak :  1.4569287251461849
Banyaknya operasi perhitungan rumus Euclidean :  500000.0
Waktu Eksekusi :  0.18829751014709473 detik

-------------------------------------------------------------[[ALGORITMA DIVIDE AND CONQUER]]-------------------------------------------------------------
Pasangan titik terdekat :  ((44.26009203802957, -47.44537870070915, 56.89074033502453), (45.13593367305393, -46.293717583225266, 56.71980347244153))
Nilai jarak : 1.4569287251461849
Banyaknya operasi perhitungan rumus Euclidean :  9965.784284662088
Waktu Eksekusi :  0.007199525833129883 detik
```

Gambar 4.4.3 Output kedua untuk masukan n = 1000



Gambar 4.4.4 Hasil visualisasi untuk bidang 3D kedua untuk jumlah titik 1000 buah

REFERENSI

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian3.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Divide-and-Conquer-(2022)-Bagian4.pdf

https://matplotlib.org/stable/index.html

LAMPIRAN

Link Google Drive dan Repository GitHub

a. Google Drive
   https://drive.google.com/drive/folders/1lMVzW-eoVeY2rcklW1YIgIaeFVUNcp65?usp=sharing

b. Repository Github
   https://github.com/ferindya/Tucil2_13521161_13521125.git

Checklist

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