

Tugas Kecil 2 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

Implementasi Closest Pair 3D dengan Algoritma *Divide* and *Conquer*



Disusun oleh :
13521086 Ariel Jovananda
13521104 Muhammad Zaydan A

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022/2023**

1. Penjelasan Algoritma *Divide and Conquer*

Strategi *divide and conquer* adalah metode penyelesaian masalah dengan cara memecah masalah menjadi beberapa sub-masalah yang lebih kecil dan sederhana, lalu menyelesaikan setiap sub-masalah tersebut, dan menggabungkannya kembali untuk mendapatkan solusi untuk masalah utuh. Algoritma ini umumnya digunakan untuk menyelesaikan masalah yang dapat dibagi menjadi beberapa sub-masalah serupa yang lebih kecil dan independen satu sama lain, sehingga solusi untuk setiap sub-masalah dapat digabungkan untuk menghasilkan solusi untuk masalah yang lebih besar. Strategi *divide and conquer* sering digunakan untuk masalah penggunaan struktur data rekursif seperti *dynamic list* dan *tree*.

Dalam implementasinya, strategi *divide and conquer* sangat terkait dengan algoritma rekursif. Algoritma rekursif terdiri dari dua bagian, yaitu *base case* dan rekurens. *Base case* dari algoritma rekursif merupakan kondisi di mana rekursivitas algoritma tersebut berhenti. Biasanya, *base case* dari algoritma rekursif adalah nilai awal atau kasus terkecil yang tidak dapat dibagi lagi. Sementara itu, rekurens dari algoritma rekursif adalah kondisi di mana algoritma tersebut dipanggil kembali dengan nilai masukan yang semakin mendekati *base case*. Pemanggilan dilakukan melalui fungsi, prosedur, method, atau ADT rekursif.

Namun, jika nilai masukan tidak berubah atau bahkan semakin menjauh dari *base case*, maka akan terjadi rekursi tak terbatas (*infinite recursion*) yang menyebabkan permasalahan tidak dapat diselesaikan. Oleh karena itu, dalam implementasi strategi *divide and conquer* dengan algoritma rekursif, *base case* dan rekurens harus didefinisikan dengan baik untuk menghindari *infinite recursion*.

Dalam implementasinya, strategi *divide and conquer* dapat dibagi menjadi tiga tahap yaitu, *divide*, *conquer*, dan *merge*. Seluruh tahap tersebut akan dijelaskan secara rinci di bawah ini :

1. *Divide* :

Tahap pertama dalam strategi *divide and conquer* adalah membagi permasalahan menjadi beberapa sub-permasalahan yang identik. Setiap sub-permasalahan harus memiliki ukuran yang hampir sama, independen satu sama lain, dan pembagiannya harus konsisten. Pada permasalahan yang sederhana seperti algoritma insertion sort atau merge sort, pembagian sub-permasalahan tidak terlalu kompleks dan menggunakan algoritma partisi yang disebut *easy split/hard join*. Namun, untuk permasalahan yang lebih kompleks seperti selection sort, quicksort, quickhull, atau closest pair of points, algoritma partisi yang digunakan untuk membagi permasalahan menjadi sub-permasalahan akan menjadi jauh lebih kompleks. Pada permasalahan yang lebih kompleks tersebut,

diperlukan algoritma yang telah terbukti secara matematis paling efektif untuk membagi permasalahan tersebut.

2. *Conquer* :

Proses *conquer* dalam strategi *divide and conquer* adalah tahap penyelesaian sub-permasalahan dengan cara melakukan perhitungan solusi secara langsung jika sub-permasalahan tersebut merupakan kasus dasar dari algoritma rekursif. Jika sub-permasalahan masih terlalu besar atau merupakan rekurensi dari algoritma rekursif, maka sub-permasalahan tersebut akan dibagi lagi menjadi beberapa sub-permasalahan yang lebih kecil dan diselesaikan secara rekursif. Untuk mempercepat proses rekursif, seorang pemrogram dapat melakukan beberapa manipulasi algoritma, seperti memoisasi atau pencatatan hasil dari algoritma rekursif yang telah dilakukan sebelumnya, atau manipulasi aljabar untuk mengurangi jumlah sub-permasalahan yang diteruskan ke dalam rekurensi. Contoh algoritma yang menggunakan strategi *conquer* adalah *binary exponentiation* di mana pemrogram hanya perlu melakukan satu kali rekurensi untuk dua sub-permasalahan yang berbeda.

3. *Merge* :

Mengkombinasikan hasil solusi dari setiap algoritma rekursif dengan masukan sub-persoalan untuk membentuk solusi utuh untuk menyelesaikan permasalahan. Seperti pada tahap *divide*, terdapat beberapa algoritma penggabungan dengan tingkat kompleksitas yang berbeda-beda, tergantung pada jenis algoritmanya. Pada tahap ini, terdapat pola menarik di mana strategi *divide and conquer* dengan algoritma partisi yang cukup sederhana akan memiliki algoritma penggabungan yang cukup kompleks. Pola ini ditemukan pada algoritma sederhana seperti *merge sort* dan *insertion sort*, di mana algoritma penggabungan dari solusi sub-persoalan menjadi kompleks. Sebaliknya, strategi *divide and conquer* dengan algoritma partisi yang kompleks akan memiliki algoritma penggabungan yang lebih mudah.

Seperti yang telah dijelaskan sebelumnya, strategi *divide and conquer* menggunakan algoritma rekursif untuk menyelesaikan permasalahan. Algoritma rekursif tersebut akan memanggil dirinya sendiri sebagai solusi dari masukkan yang diberikan, sehingga muncul permasalahan baru ketika ingin menganalisis kompleksitas waktu dari algoritma tersebut. Karena rekursivitas algoritmanya, kompleksitas waktu algoritma tersebut juga memiliki bentuk rekursif, yang membuat penulis kesulitan untuk menentukan notasi Big O dan mengevaluasi efektivitas strategi *divide and conquer*. Oleh karena itu, penulis akan menggunakan Teorema Master untuk membantu menentukan kompleksitas waktu dari algoritma rekursif. Penjelasan mengenai Teorema Master dapat ditemukan di bawah ini.

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 :

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

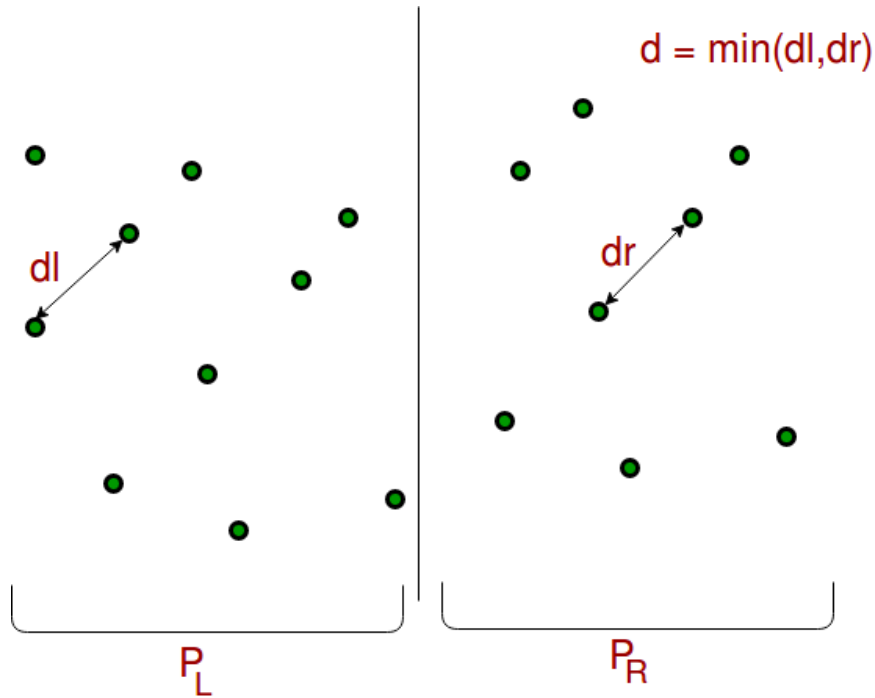
Permasalahan yang diberikan untuk tugas kecil ini adalah diberikan himpunan titik, P, yang terdiri dari n buah titik pada bidang 3-D, (x_i, y_i, z_i) , $i = 1, 2, \dots, n$. Tentukan sepasang titik di dalam P yang jaraknya terdekat satu sama lain. Untuk point (x_i, y_i, z_i) menggunakan tipe double.

Berikut ini adalah penjelasan langkah-langkah dari Algoritma *divide and conquer* untuk memecahkan persoalan *Closest Pair 3-D* :

1. Urutkanlah titik-titik di dalam p berdasarkan absisnya (x) menggunakan algoritma sort yang telah diimplementasikan sendiri.

***DIVIDE* :**

2. Bagi menjadi dua titik-titik tersebut berdasarkan titik tengah absis, titik-titik akan dibagi menjadi dua larik yang bernama *left* dan *right*
3. Urutkan titik-titik di dalam p berdasarkan ordinatnya (y) menggunakan algoritma sort yang telah diimplementasikan sendiri.



Gambar 1 Gambaran Membagi dua larik

Sumber : <https://media.geeksforgeeks.org/wp-content/uploads/mindis.png>

4. Ulangi proses tersebut sampai titik-titik sisa kurang dari atau sama dengan 3 dengan proses rekursif

CONQUER :

5. Jika titik-titik sudah sisa kurang dari atau sama dengan 3, lakukan *brute force* pada perhitungan *euclidean distance* untuk semua kemungkinan pasangan dan mengambil nilai *euclidean distance* yang paling kecil
6. Ulangi tahap 4 dengan proses rekursif pada larik partisi *left* dan *right*

COMBINE :

7. Bandingkan nilai *euclidean distance* yang didapatkan dari larik *left* dan *right*, dan simpan nilai *euclidean distance* yang paling kecil antara kedua larik

2. Source Code dengan bahasa python

a. visualizer.py

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def visualizer(points, pair): #Visualisasi Grafik Titik-Titik
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    xs = [p[0] for p in points]
    ys = [p[1] for p in points]
    zs = [p[2] for p in points]
    ax.scatter(xs, ys, zs)

    if pair:
        x_pair = [pair[0][0], pair[1][0]]
        y_pair = [pair[0][1], pair[1][1]]
        z_pair = [pair[0][2], pair[1][2]]
        ax.scatter(x_pair, y_pair, z_pair, c='red', s=80) #Mengubah warna
titik yang merupakan closestPair

    #Memberi Label
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    plt.show()
```

b. ClosestPair.py

```
import math
import random

count = 0; #global variable
class Point: #class point (x,y,z)
    def __init__(self, x, y, z):
        self.x = x
```

```

        self.y = y
        self.z = z

    def __repr__(self):
        return f'({self.x}, {self.y}, {self.z})'

def generatePoints(row, col):
    matrix = [[random.uniform(0,100) for j in range(col)] for i in
range(row)]
    return matrix

def distance(point1, point2):
    sum = 0
    for i in range(len(point1)):
        sum += ((point1[i] - point2[i]) ** 2)
    result = math.sqrt(sum)
    return result

def sortList(mat, axis):
    if len(mat) <= 1:
        return mat
    acuan = mat[0]
    left = []
    right = []
    for i in mat[1:]:
        if i[axis] < acuan[axis]:
            left.append(i)
        else:
            right.append(i)
    return sortList(left,axis) + [acuan] + sortList(right,axis)

def bruteForce(List):
    global count
    n=len(List)
    min=float('inf')
    cp=None

    for i in range(n):
        for j in range(i+1,n):
            temp=distance(List[i],List[j])

```

```

        count+=1
        if temp < min :
            min=temp
            cp= [List[i],List[j]]
    return min , cp

def closestPairRec(List):
    global count
    n=len(List)
    if n<=3 :
        return bruteForce(List)

    mid= n//2
    midP=List[mid]

    left_p=List[:mid]
    right_p=List[mid:]

    min_l,cp_l=closestPairRec(left_p)
    min_r,cp_r=closestPairRec(right_p)

    minP=min(min_l,min_r)
    cp=cp_r if min_r<min_l else cp_l
    unsorted=[]
    for point in List:
        if abs(point[0] - midP[0])<minP :
            unsorted.append(point)

    sorted = sortList(unsorted, 1)

    for i in range(len(sorted)):
        j=i+1
        while j < len(sorted) and sorted[j][1]-sorted[i][1] < minP :
            tempMin=distance(sorted[i],sorted[j])
            count += 1
            if tempMin < minP:
                minP=tempMin
                cp=[sorted[i],sorted[j]]
            j+=1

```



```

#generate random point
row = int(input("Masukkan jumlah point\t: "))
col = int(input("Masukkan dimensi\t: "))
if col == 3:
    stbf = time.time()
    ListPoint = cp.generatePoints(row,col)
    print(f"{Fore.RED}Pure Brute Force{Fore.WHITE}")
    Result1 = cp.bruteForce(ListPoint)
    print(Result1)
    print('EucDistance count\t:', cp.count)
    RunTime1 = (time.time() - stbf)
    print('Execution time\t\t:', RunTime1, 'seconds'," (ROG-G513qr)")
    print(f"{Fore.RED}Divide and Conqeur{Fore.WHITE}")
    st = time.time()
    cp.count = 0
    Result = cp.closestPair(ListPoint)
    print(Result)
    print('EucDistance count\t:', cp.count)
    RunTime = (time.time() - st)
    print('Execution time\t\t:', RunTime, 'seconds'," (ROG-G513qr)")
    pil = str(input("Apakah ingin divisualisasikan? \nTekan tombol
selain Y/y jika tidak\n"))
    if(pil == "Y" or pil == "y"):
        vis.visualizer(ListPoint, Result[1])
    else:
        return 0
else:
    stbf = time.time()
    ListPoint = cp.generatePoints(row,col)
    print(f"{Fore.RED}Pure Brute Force{Fore.WHITE}")
    Result1 = cp.bruteForce(ListPoint)
    print(Result1)
    print('EucDistance count\t:', cp.count)
    RunTime1 = (time.time() - stbf)
    print('Execution time\t\t:', RunTime1, 'seconds'," (ROG-G513qr)")
    print(f"{Fore.RED}Divide and Conqeur{Fore.WHITE}")
    st = time.time()
    cp.count = 0
    Result = cp.closestPair(ListPoint)
    print(Result)

```

```

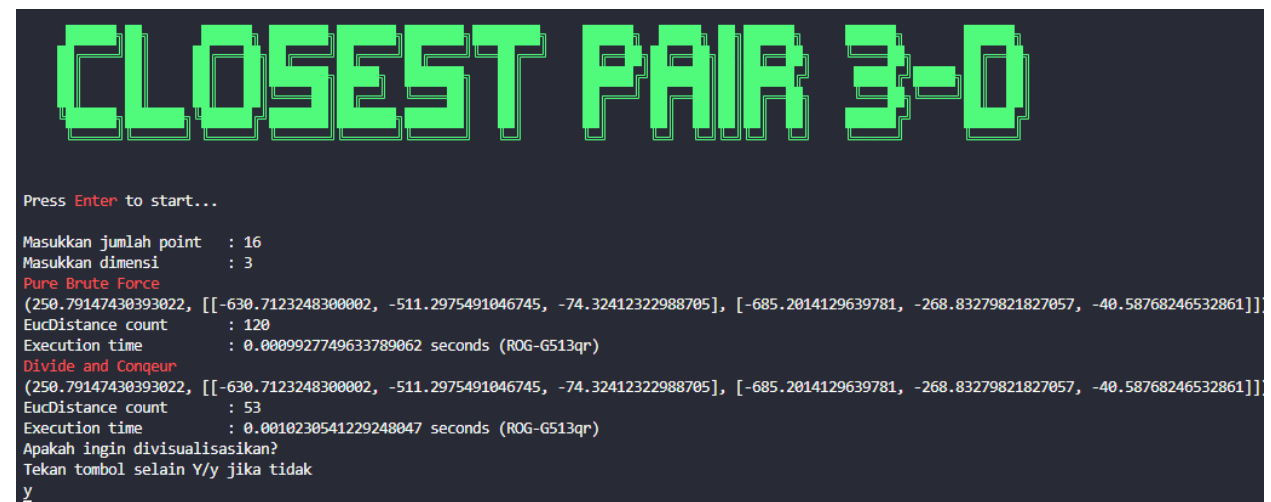
print('EucDistance count\t:', cp.count)
RunTime = (time.time() - st)
print('Execution time\t\t:', RunTime, 'seconds', "(ROG-G513qr)")
return 0

main()

```

3. *Screenshot* untuk $n = 16$, $n = 64$, $n = 128$, $n = 1000$

a. $n=16$, Dimensi=3



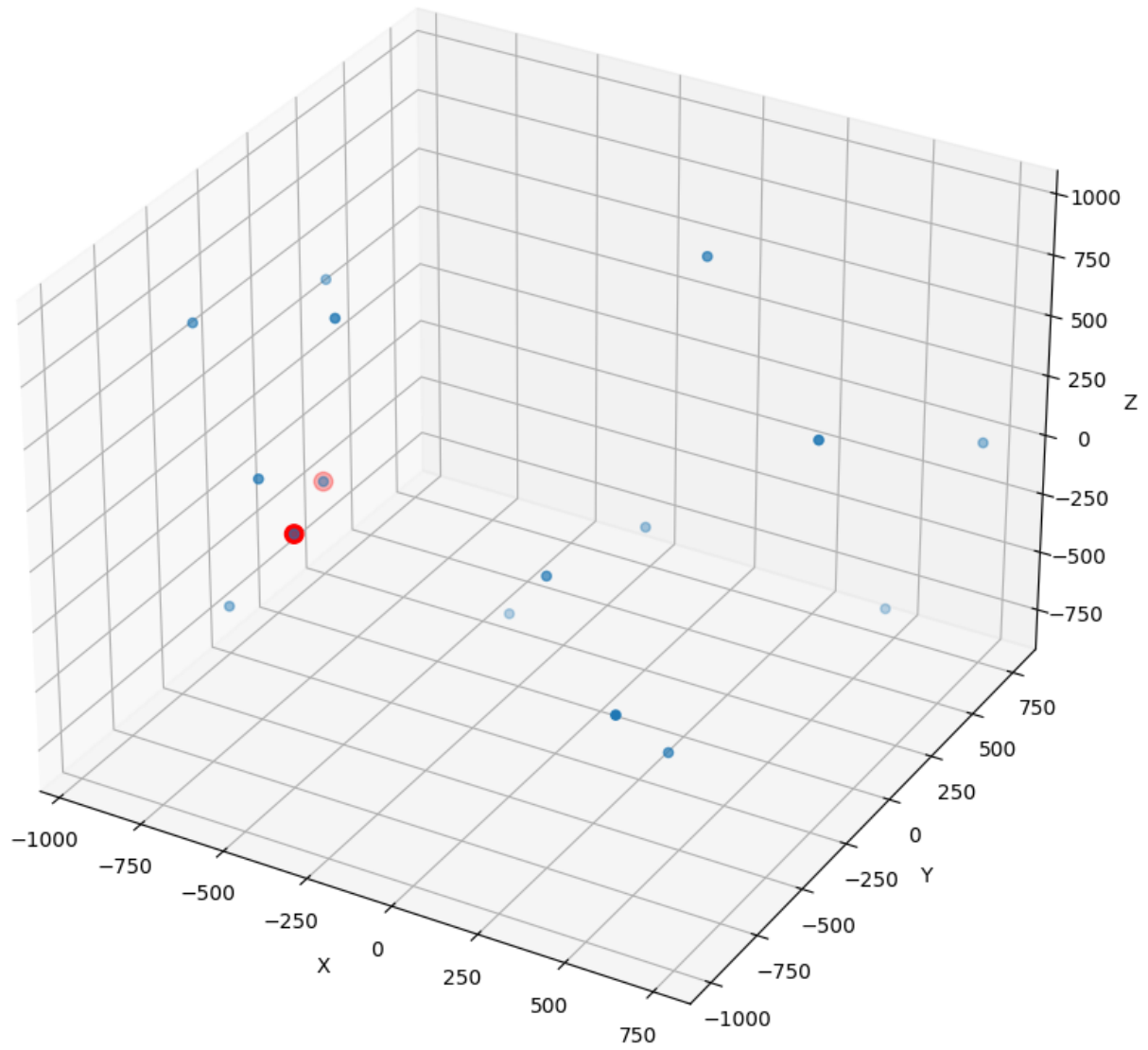
```

CLOSEST PAIR 3-D

Press Enter to start...

Masukkan jumlah point : 16
Masukkan dimensi      : 3
Pure Brute Force
(250.79147430393022, [[-630.7123248300002, -511.2975491046745, -74.32412322988705], [-685.2014129639781, -268.83279821827057, -40.58768246532861]])
EucDistance count      : 120
Execution time          : 0.0009927749633789062 seconds (ROG-G513qr)
Divide and Conquer
(250.79147430393022, [[-630.7123248300002, -511.2975491046745, -74.32412322988705], [-685.2014129639781, -268.83279821827057, -40.58768246532861]])
EucDistance count      : 53
Execution time          : 0.0010230541229248047 seconds (ROG-G513qr)
Apakah ingin divisualisasikan?
Tekan tombol selain Y/y jika tidak
y

```



b. $n=64$, Dimensi=3

CLOSEST PAIR 3-D

Press **Enter** to start...

Masukkan jumlah point : 64

Masukkan dimensi : 3

Pure Brute Force

(110.010904407105, [[-961.8548835601872, -190.68133044127183, -308.2424523681167], [-887.0084453854339, -168.97594406585824, -385.8909293867681]]])

EucDistance count : 2016

Execution time : 0.005517482757568359 seconds (ROG-G513qr)

Divide and Congeur

(110.010904407105, [[-961.8548835601872, -190.68133044127183, -308.2424523681167], [-887.0084453854339, -168.97594406585824, -385.8909293867681]]])

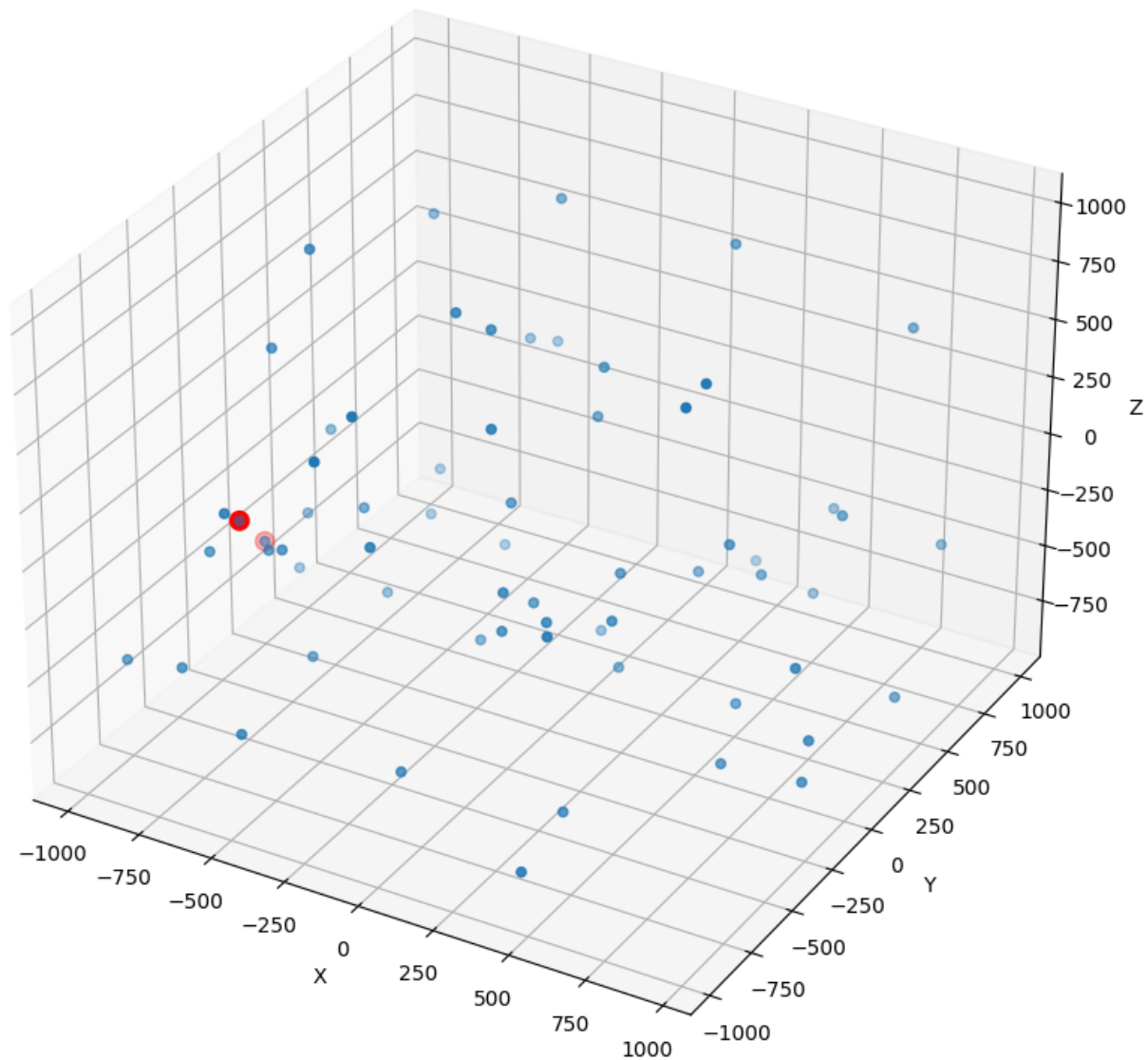
EucDistance count : 164

Execution time : 0.001659154891967734 seconds (ROG-G513qr)

Apakah ingin divisualisasikan?

Tekan tombol selain Y/y jika tidak

y

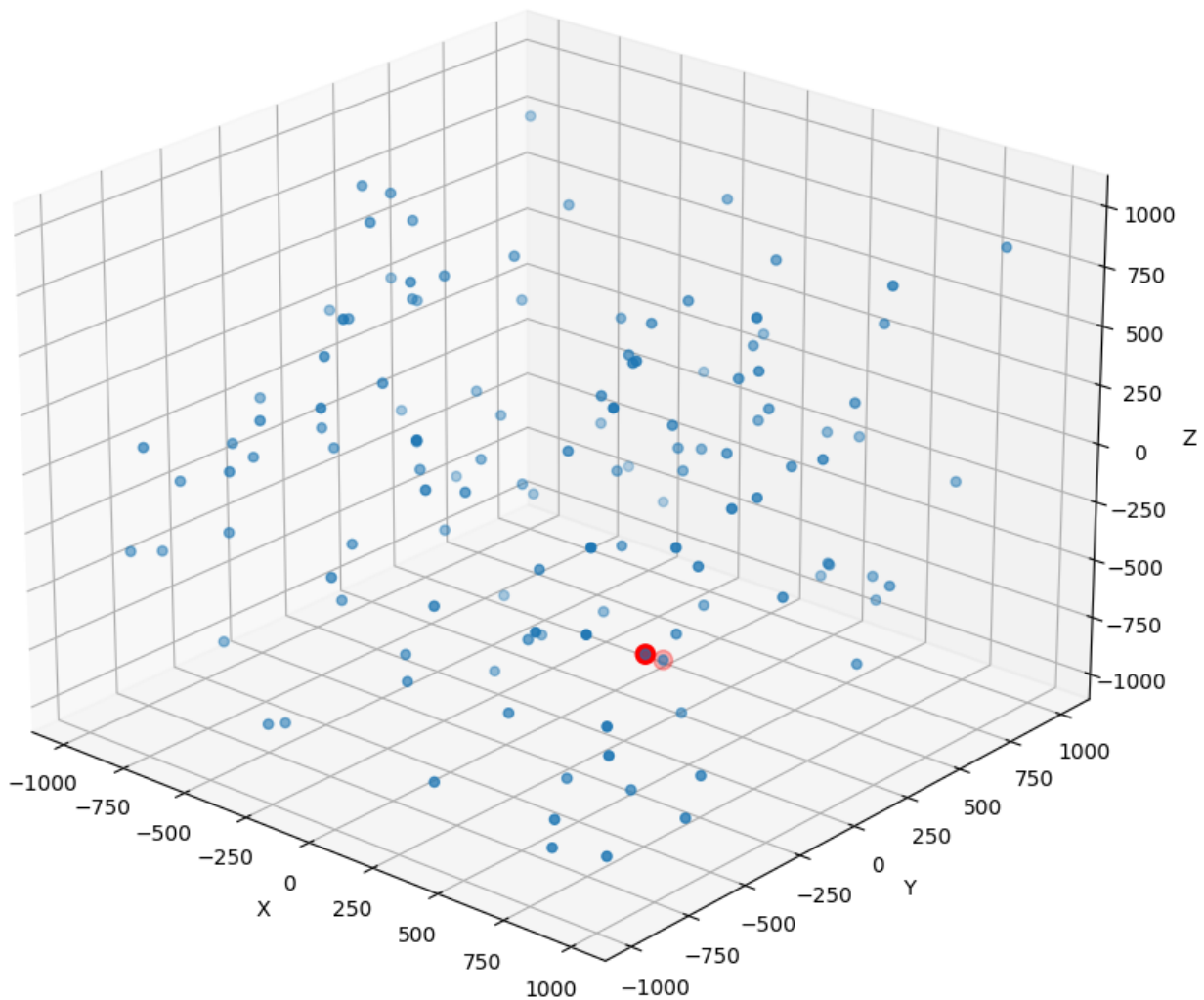


c. $n=128, \text{Dimensi}=3$

```
CLOSEST PAIR 3-D

Press Enter to start...

Masukkan jumlah point : 128
Masukkan dimensi : 3
Pure Brute Force
(60.88595082729092, [[398.3100609632513, 0.8047899100780569, -697.8995984125459], [363.90548322889754, -39.73886952803741, -668.2409450643204]])
EucDistance count : 8128
Execution time : 0.010897636413574219 seconds (ROG-G513qr)
Divide and Conquer
(60.88595082729092, [[363.90548322889754, -39.73886952803741, -668.2409450643204], [398.3100609632513, 0.8047899100780569, -697.8995984125459]])
EucDistance count : 338
Execution time : 0.0023314952850341797 seconds (ROG-G513qr)
Apakah ingin divisualisasikan?
Tekan tombol selain Y/y jika tidak
█
```



d. $n=1000, \text{Dimensi}=3$

CLOSEST PAIR 3-D

Press **Enter** to start...

Masukkan jumlah point : 1000

Masukkan dimensi : 3

Pure Brute Force

(11.290913471895355, [[-367.8992509000092, 243.43462117174136, 772.7313719229396], [-376.58773513215203, 250.62061266122942, 773.3284440212981]])

EucDistance count : 499500

Execution time : 0.5652129650115967 seconds (ROG-G513qr)

Divide and Congeur

(11.290913471895355, [[-367.8992509000092, 243.43462117174136, 772.7313719229396], [-376.58773513215203, 250.62061266122942, 773.3284440212981]])

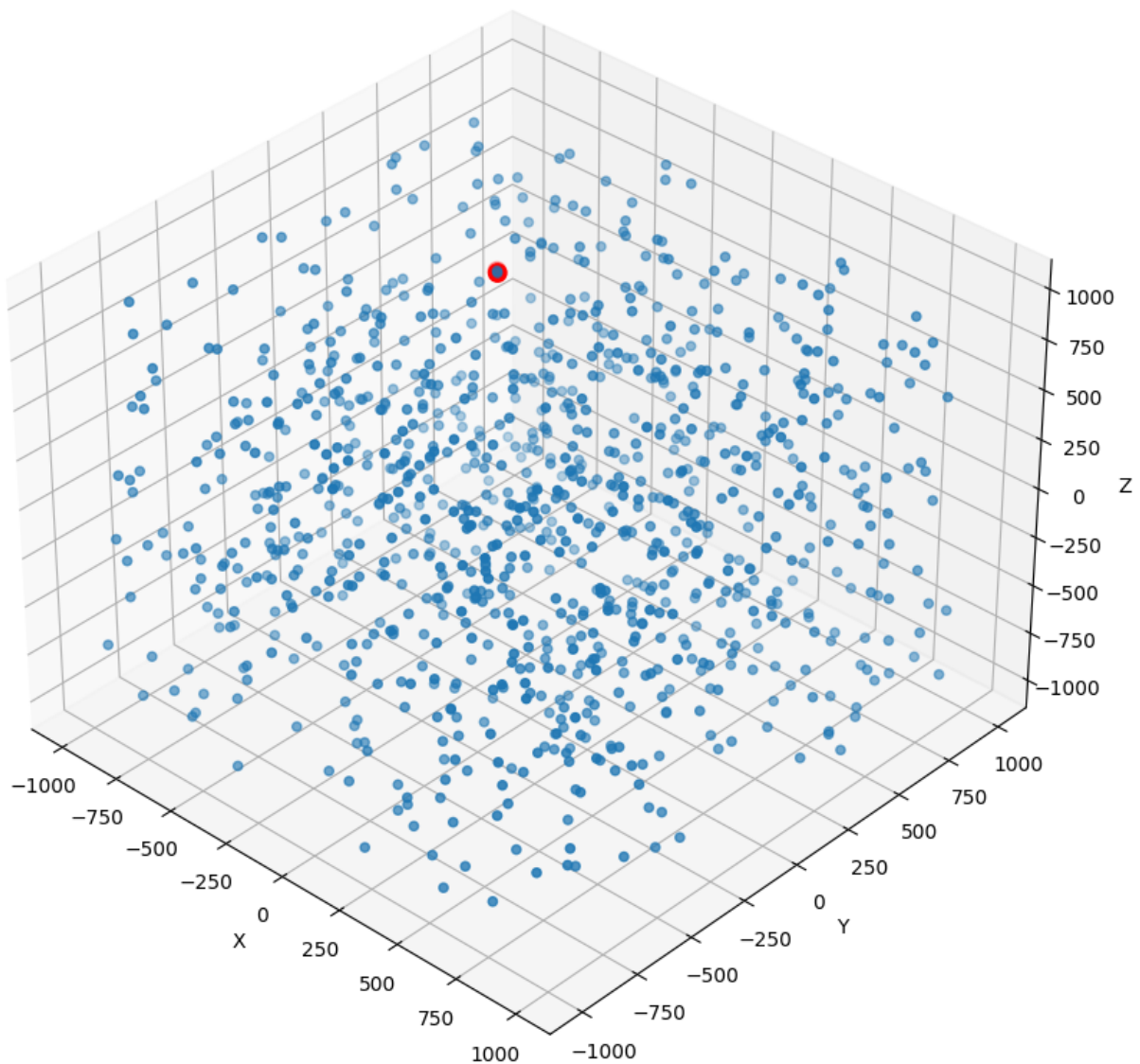
EucDistance count : 4617

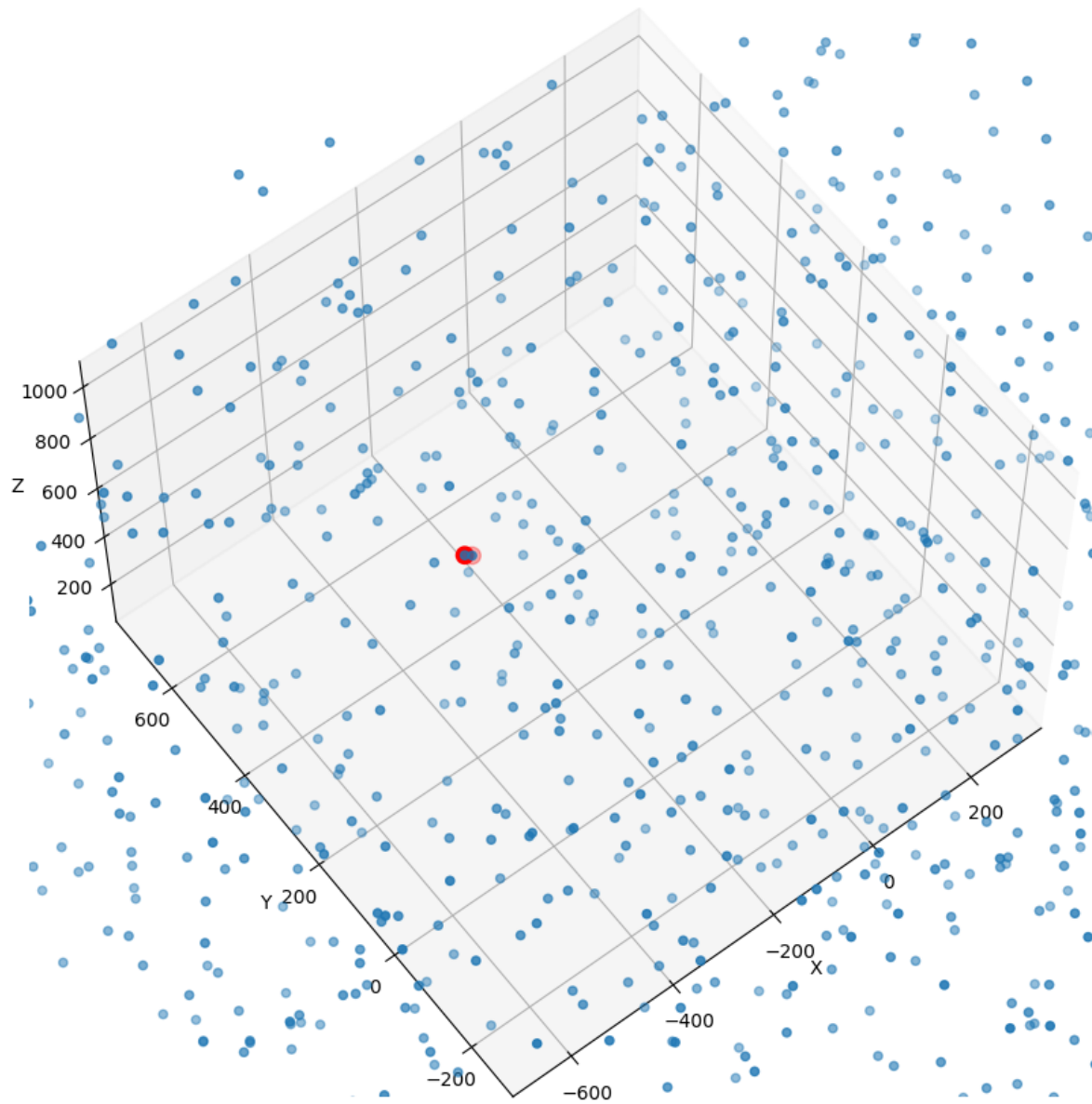
Execution time : 0.0238802433013916 seconds (ROG-G513qr)

Apakah ingin divisualisasikan?

Tekan tombol selain Y/y jika tidak

y





e. Bonus 2 , n=16 ,Dimensi=2


```
CLOSEST PAIR 3-0

Press Enter to start...

Masukkan jumlah point : 16
Masukkan dimensi      : 2
Pure Brute Force
(42.677079565602035, [[-970.8641273289478, 47.106577541010665], [-938.995500702374, 18.72130394386602]])
EucDistance count     : 120
Execution time        : 0.00099945068359375 seconds (ROG-G513qr)
Divide and Conquer
(42.677079565602035, [[-970.8641273289478, 47.106577541010665], [-938.995500702374, 18.72130394386602]])
EucDistance count     : 11
Execution time        : 0.001051187515258789 seconds (ROG-G513qr)
```

f. Bonus 2 , n=16 ,Dimensi=1

```
CLOSEST PAIR 3-0

Press Enter to start...

Masukkan jumlah point : 16
Masukkan dimensi      : 1
Pure Brute Force
(4.636699423966775, [[865.7837961808575], [861.1470967568907]])
EucDistance count     : 120
Execution time        : 0.0010194778442382812 seconds (ROG-G513qr)
Divide and Conquer
(4.636699423966775, [[861.1470967568907], [865.7837961808575]])
EucDistance count     : 8
Execution time        : 0.0 seconds (ROG-G513qr)
PS C:\Users\ASUS\Tucil2Stima\Tucil2_13521086_13521104> █
```

4. Link Repository Github

https://github.com/arieljovananda88/Tucil2_13521086_13521104

5. Checklist

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa ada kesalahan.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2. Program berhasil running	<input checked="" type="checkbox"/>	<input type="checkbox"/>

3. Program dapat menerima masukan dan dan menuliskan luaran.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4. Luaran program sudah benar (solusi closest pair benar)	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5. Bonus 1 dikerjakan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5. Bonus 2 dikerjakan	<input checked="" type="checkbox"/>	<input type="checkbox"/>