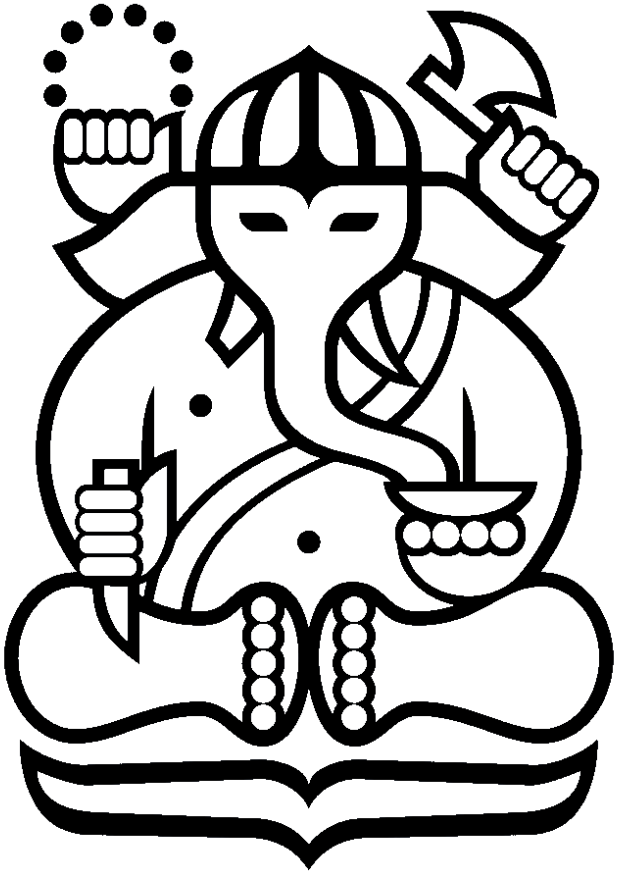
TUGAS KECIL 2 IF2211 STRATEGI ALGORITMA
SEMESTER II TAHUN 2022/2023
Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer



Disusun oleh :
13521054 - Wilson Tansil
13521127 - Marcel Ryan Antony

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

DAFTAR ISI

BAB I

DESKRIPSI MASALAH

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 titk $P_1 = (x_1, y_1, z_1)$ dan $P_2 = (x_2, y_2, z_2)$ 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*.

BAB II

LANDASAN TEORI

2.1 Algoritma Divide and Conquer

Di dalam ilmu kompute, algoritma *divide and conquer* merupakan salah satu algoritma paling populer dikarenakan memiliki kompleksitas waktu yang efisien dalam sebagian kasus. Algoritma *divide and conquer* awalnya merupakan strategi militer dalam memecah belah musuh dari bagian internal musuh itu sendiri. Strategi politik tersebut akan memecah (*divide*) sebuah musuh menjadi beberapa bagian kecil yang tidak bersatu dan menembus (*conquer*) satu persatu dengan lebih cepat dan mudah. Maka dari itu, algoritma yang berdasarkan strategi tersebut juga terbagi menjadi dua bagian yakni *divide* dan *conquer*. *Divide* dalam kasus ini adalah membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil, sedangkan *conquer* dalam kasus ini bertugas untuk menyelesaikan masing-masing upa-persoalan. Tidak hanya itu algoritma *divide and conquer* juga dibantu dengan *combine* yang menggabungkan solusi masing-masing persoalan sehingga membentuk solusi persoalan semula.

```
procedure DIVIDEandCONQUER(input P : problem, n : integer)
{ Menyelesaikan persoalan P dengan algoritma divide and conquer
  Masukan: masukan persoalan P berukuran n
  Luaran: solusi dari persoalan semula   }
Deklarasi
   r : integer

Algoritma
  if n ≤ n0 then  {ukuran persoalan P sudah cukup kecil }
     SOLVE  persoalan P yang berukuran n ini
  else
     DIVIDE  menjadi r upa-persoalan, P1, P2, …, Pr , yang masing-masing berukuran n1, n2, …, nr
     for  masing-masing P1, P2, …, Pr , do
        DIVIDEandCONQUER(Pi,  ni)
     endfor
     COMBINE  solusi dari P1, P2, …, Pr  menjadi solusi persoalan semula
  endif
```

Gambar 2.1.1 Skema Umum Algoritma *Divide and Conquer*

2.2 Pasangan Titik Terdekat 3D

Mencari pasangan titik terdekat dalam ruang berdimensi tiga merupakan tugas utama dalam tugas kecil kali ini. Seperti yang telah kita pelajari sejak kecil, sebuah objek dikatakan berdimensi tiga apabila orang tersebut memiliki lebar, tinggi, dan tebal yang apabila kita melihat dari sudut pandang bidang kartesius dapat direpresentasikan sebagai sumbu x, y, dan z. Dalam kasus ini kita diberikan tugas untuk menentukan pasangan titik terdekat dalam dimensi tiga dari n titik sesuai keinginan pengguna. Jarak dari dua titik dapat kita tentukan dengan menggunakan rumus

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

Untuk mendapatkan jarak minimum dari kedua titik, algoritma *divide and conquer* diimplementasikan dengan tujuan memberikan hasil yang akurat, cepat, dan memiliki tingkat stabilitas yang tinggi untuk pertumbuhan jumlah data. Algoritma *brute force* juga ikut diimplementasikan sebagai variabel perbandingan dengan algoritma *divide and conquer.* Dalam hal ini, yang diperbandingkan adalah waktu dan akurasi dari hasil penerapan *divide and conquer* dan hasil penerapan *brute force* yang dianggap relatif benar.

2.3 Algoritma Penyelesaian Pasangan Titik Terdekat 3D dengan Pendekatan Divide and Conquer

Dalam menyelesaikan permasalahan *closest pair* pada dimensi *3D*, penulis menggunakan pendekatan algoritma *divide and conquer*. Berikut adalah langkah-langkah penyelesaian permasalahan dengan algoritma dijelaskan secara deskriptif :

1. Lakukan *sorting* pada kumpulan titik berdasarkan sumbu x menaik
2. Setelah dilakukan *sorting* bagi kumpulan titik menjadi 2 bagian yaitu bagian kiri yang nilai x nya lebih kecil dari median dan bagian kanan yang nilai x nya lebih besar dari median
3. Panggil kembali algoritma *divide and conquer* secara rekursif untuk himpunan kiri dan himpunan kanan untuk mencari *closest pair* diantara kedua himpunan tersebut
4. Basis rekursif pada algoritma *divide and conquer* adalah ketika himpunan titik hanya berisi 2 atau 3 titik. Pada basis, pencarian *closest pair* menggunakan algoritma *brute force* karena titik tidak banyak sehingga tidak memakan banyak waktu
5. Setelah didapat *closest pair* untuk himpunan kiri dan himpunan kanan, bandingkan keduanya dan ambil *closest pair* yang memiliki distance lebih rendah
6. Kemudian harus dipertimbangkan lagi apabila ada titik pada himpunan kiri dan titik pada himpunan kanan yang kemungkinan jarak antar keduanya lebih kecil dari jarak minimum yang diperoleh dari kedua himpunan
7. Titik-titik yang dicek adalah titik-titik yang berada di sekitar median dan memenuhi syarat berupa jarak sumbu-sumbu pada titik tersebut dengan sumbu-sumbu titik median lebih kecil dari nilai minimum sekarang
8. Setelah didapatkan kumpulan titik-titik yang perlu di cek pada sekitar median, carilah nilai minimum antara kumpulan titik-titik tersebut.
9. Setelah didapatkan minimum dari kumpulan titik-titik pada sekitar median, bandingkan dengan nilai minimum sekarang. Apabila minimum tengah lebih besar maka ambil *closest pair* pada titik sekitar median dan sebaliknya jika tidak.
10. Visualisasikan hasil yang telah didapat.

BAB III

IMPLEMENTASI ALGORITMA DALAM BAHASA PYTHON

3.1 closestPoint.py

File ini berisi pustaka closestPoint yang berguna untuk mencari pasangan titik terdekat dari kumpulan-kumpulan titik. Beberapa fungsi yang terdapat pada pustaka adalah sebagai berikut :

Function	Description
inputUser()	Fungsi ini digunakan untuk menerima input dari user berupa dimensi dan berapa titik yang ingin di generate
partition(array, low, high)	Fungsi ini digunakan sebagai fungsi pembantu untuk quicksort yang mempartisi array menjadi 2 array yaitu array yang isinya lebih kecil dari pivot dan array yang isinya lebih besar dari pivot
quickSort(array, low, high)	Fungsi ini digunakan untuk mengurutkan array of points berdasarkan sumbu x
minDistanceBruteForce(array, dimensi)	Fungsi ini digunakan untuk mencari pasangan titik terdekat dengan menggunakan algoritma *brute force*
distance(point1, point2, dimensi)	Fungsi ini digunakan untuk mencari euclidean distance antara 2 titik
needToCheck(point1, point2, minimum, dimensi)	Fungsi ini digunakan untuk mengecek apakah dua buah titik perlu diproses lebih lanjut pada strip di dimensi diatas 3. Sebagai contoh apabila jarak 2 titik pada sumbu x dan sumbu y telah melebihi min^2 maka titik tersebut sudah tidak perlu di proses lagi
divideAndConquer(points, dimensi)	Fungsi ini digunakan untuk mencari pasangan titik terdekat dengan menggunakan algoritma *divide and conquer*
jumlahEucDNC()	Fungsi ini digunakan untuk mengeluarkan output berupa jumlah operasi perhitungan euclidean distance pada algoritma *divide and conquer*
indexPoint(array, points)	Fungsi ini digunakan untuk mencari titik berada pada index ke berapa pada array of points

Program closestPoint.py

```
import random
import math
import sys
import time
sys.setrecursionlimit(10000)

def inputUser(choice):
    n = int(input("Masukkan berapa titik yang ingin di generate: "))
    dimensi = int(input("Ingin berapa dimensi: "))
    while (n <= 0 or dimensi <= 0):
        print("Input tidak valid!, Silahkan input jumlah titik dan
dimensi dengan nilai diatas 0!")
        n = int(input("Masukkan berapa titik yang ingin di generate:
"))
        dimensi = int(input("Ingin berapa dimensi: "))
    array = []
    if(choice == 1):
            # Making random always different every time it runs
        random.seed(time.time())
        for i in range(n):
            points = []
            for j in range(dimensi):
                randomvalue = random.uniform(0, 100)
                points.append(randomvalue)
            array.append(points)
    elif(choice == 2):
        print("Every component x,y,z seperated by comma")
        for i in range(n):
            inputStr = input("Input point {}: ".format(i+1))
            inputStrList = inputStr.split(",")
            points = [int(x) for x in inputStrList]
            array.append(points)

    # Mengurutkan array dari x axis menaik
    return array, dimensi

def partition(array, low, high):
    pivot = array[high][0]
    i = low - 1
```

```
    for j in range(low, high):
        if array[j][0] <= pivot:
            i = i + 1
            (array[i], array[j]) = (array[j], array[i])
    (array[i + 1], array[high]) = (array[high], array[i + 1])
    return i + 1

def quickSort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quickSort(array, low, pi - 1)
        quickSort(array, pi + 1, high)

def minDistanceBruteForce(array, dimensi):
    count = 0
    start = time.time()
    array_distance = []
    for i in range(len(array)):
        temp = []
        for j in range(i+1, len(array)):
            jarak = 0
            for k in range(dimensi):
                jarak += (array[j][k] - array[i][k]) ** 2
            temp.append(math.sqrt(jarak))
            count += 1
        array_distance.append(temp)
    min = array_distance[0][0]
    point1= 0
    point2 = 0
    for i in range(len(array_distance)):
        for j in range(len(array_distance[i])):
            if (min > array_distance[i][j]):
                min = array_distance[i][j]
                point1 = i
                point2 = i+j+1
    end = time.time()
    print("Dengan menggunakan algoritma brute force didapat hasil
sebagai berikut :")
    print("Titik 1 : (", end='')
    for i in range(dimensi):
```

```
        if (i == dimensi - 1):
            print(f"{array[point1][i]:.3f})")
        else:
            print(f"{array[point1][i]:.3f}, ", end= '')
    print("Titik 2 : (", end='')
    for i in range(dimensi):
        if (i == dimensi - 1):
            print(f"{array[point2][i]:.3f})")
        else:
            print(f"{array[point2][i]:.3f}, ", end= '')
    print(f"Waktu : {(end-start) * 1000:.3f} ms")
    print(f"Jumlah operasi euclidean distance untuk algoritma brute
force : {count}")
    print(f"Jarak antar titik : {min:.3f}")
    return point1+1, point2+1

# Variabel untuk menghitung operasi euclidean distance
count = 0
def distance(point1, point2, dimensi):
    global count
    jarak = 0
    for i in range(dimensi):
        jarak += (point2[i] - point1[i]) ** 2
    count += 1
    return math.sqrt(jarak)


# Function to check whether point1 and point2 needs to be checked
further
def needToCheck(point1, point2, minimum, dimensi):
    proses = True
    jarak = 0
    for i in range(dimensi):
        jarak += (point2[i] - point1[i]) ** 2
        if (jarak > minimum**2):
            proses = False
            break
    return proses

def divideAndConquer(points, dimensi):
```

```
    # Mengurutkan array dari x axis menaik
    quickSort(points, 0, len(points) - 1)
    array = []
    if (len(points) == 2):
        array = points
        min = distance(points[0], points[1], dimensi)
        return min, array
    elif (len(points) == 3):
        min1 = distance(points[0], points[1], dimensi)
        min2 = distance(points[0], points[2], dimensi)
        min3 = distance(points[1], points[2], dimensi)
        if (min1 <= min2 and min1 <= min3):
            array = [points[0], points[1]]
            return min1, array
        elif (min2 <= min1 and min2 <= min3):
            array = [points[0], points[2]]
            return min2, array
        else:
            array = [points[1], points[2]]
            return min3, array
    else:
        mid = len(points)//2
        left = []
        right = []
        # Membagi titik menjadi dua himpunan kanan kiri. Dengan syarat
titik sudah terurut menaik berdasarkan sumbu x
        for i in range(mid):
            left.append(points[i])
        for i in range(mid, len(points)):
            right.append(points[i])
        minLeft, arrayLeft = divideAndConquer(left, dimensi)
        minRight, arrayRight = divideAndConquer(right, dimensi)
        if (minLeft < minRight):
            min = minLeft
            array = arrayLeft
        else:
            min = minRight
            array = arrayRight
```

```
        # Kondisi dimana ada titik pada himpunan kiri dan kanan yang
jaraknya kemungkinan lebih kecil dari minLeft atau minRight

        midPoint = []
        # Mengambil titik yang jarak x nya lebih kecil atau sama dengan
min dari titik x median
        for i in range(len(points)):
            if (abs(points[i][0] - points[mid][0]) <= min):
                midPoint.append(points[i])

        # Mencari pasangan titik yang nilai distancenya lebih kecil
dari minLeft atau minRight jika ada
        for i in range(len(midPoint)):
            for j in range(i+1, len(midPoint)):
                if (dimensi > 3):
                    if (needToCheck(midPoint[i], midPoint[j], min,
dimensi)):
                        minMid = distance(midPoint[i], midPoint[j],
dimensi)
                        if (minMid < min):
                            min = minMid
                            array = [midPoint[i], midPoint[j]]
                else:
                    proses = True
                    for k in range(dimensi):
                        if (abs(midPoint[i][k] - midPoint[j][k]) >
min):
                            proses = False
                            break
                    if (proses):
                        minMid = distance(midPoint[i], midPoint[j],
dimensi)
                        if (minMid < min):
                            min = minMid
                            array = [midPoint[i], midPoint[j]]
        return min, array

def jumlahEucDNC():
    print("Jumlah operasi euclidean distance untuk algoritma divide and
conquer :", count)
```

```
# Searching index of the point in array of points
def indexPoint(array, points):
    for i in range(len(points)):
        if (points[i] == array[0]):
            idx1 = i
            break
    for i in range(len(points)):
        if (points[i] == array[1]):
            idx2 = i
            break
    return idx1, idx2
```

3.2 visualization.py

File ini berisi kode program yang digunakan untuk visualisasi dan juga sebagai *driver* untuk penggunaan fungsi-fungsi yang ada pada closestPoint.py

```
import matplotlib.pyplot as plt
import closestPoint
import time
import os

os.system('cls')
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')
print("Selamat datang di program mencari pasangan titik terdekat pada
dimensi ke-n !")
print("Input Process")
print(
    """
There is 2 ways to input
1. Random
2. Manual
Please include the number and it is not validated so recheck before you
press enter!
    """
)
```

```
inputChoice = int(input("How do you wanna input : "))
while (inputChoice > 2 or inputChoice < 1):
    print("Input tidak valid silakan input pilihan antara 1 dan 2
saja")
    inputChoice = int(input("How do you wanna input : "))
array_of_points, dimensi = closestPoint.inputUser(inputChoice)
point1, point2 = closestPoint.minDistanceBruteForce(array_of_points,
dimensi)
start = time.time()
mindnc, arraydnc  = closestPoint.divideAndConquer(array_of_points,
dimensi)
end = time.time()
idx1, idx2 = closestPoint.indexPoint(arraydnc, array_of_points)
print('')
print("Dengan menggunakan algoritma divide and conquer didapat hasil
sebagai berikut :")
print("Titik 1: (", end='')
for i in range(dimensi):
    if (i == dimensi - 1):
        print(f"{array_of_points[idx1][i]:.3f})")
    else:
        print(f"{array_of_points[idx1][i]:.3f}, ", end= '')
print("Titik 2 : (", end='')
for i in range(dimensi):
    if (i == dimensi - 1):
        print(f"{array_of_points[idx2][i]:.3f})")
    else:
        print(f"{array_of_points[idx2][i]:.3f}, ", end= '')
print(f"Waktu : {(end-start) * 1000:.3f} ms")
closestPoint.jumlahEucDNC()
print(f"Jarak antar titik : {mindnc:.3f}")
print("Apabila terdapat perbedaan titik, namun jarak euclidean sama hal
ini mungkin terjadi apabila ada 2 pasangan titik yang memiliki jarak
euclidean yang sama")

if (dimensi == 3):
    for i in range(len(array_of_points)):
        if (i == idx1 or i == idx2):
            ax.scatter(array_of_points[i][0], array_of_points[i][1],
array_of_points[i][2], color = "red")
```

```
        else:
            ax.scatter(array_of_points[i][0], array_of_points[i][1],
array_of_points[i][2], c= 'b')


    plt.plot([array_of_points[idx1][0] ,array_of_points[idx2][0]] ,
[array_of_points[idx1][1], array_of_points[idx2][1]],
[array_of_points[idx1][2], array_of_points[idx2][2]], color = 'black',
linestyle = '-')
    plt.ioff()
    plt.title("3D Scatter Plot")
    ax.set_xlabel("Sumbu X")
    ax.set_ylabel("Sumbu Y")
    ax.set_zlabel("Sumbu Z")
    plt.show()
```

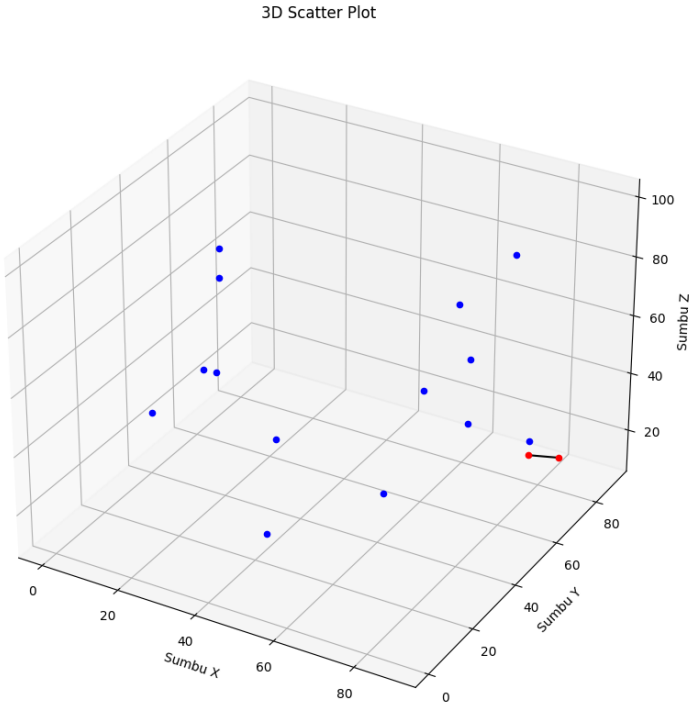
3.3 Dependencies

1. random
2. math
3. sys
4. time
5. os
6. matplotlib

BAB IV

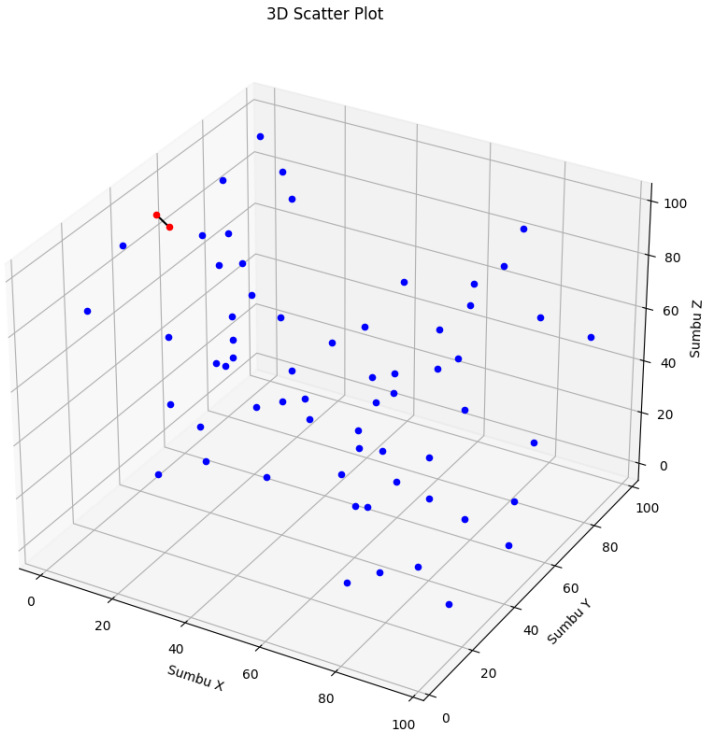
MASUKAN DAN KELUARAN PROGRAM

```
How do you wanna input : 1
Masukkan berapa titik yang ingin di generate: 16
Ingin berapa dimensi: 3
Dengan menggunakan algoritma brute force didapat hasil sebagai berikut :
Titik 1 : (88.615, 72.572, 23.713)
Titik 2 : (82.446, 69.620, 24.186)
Waktu : 1.002 ms
Jumlah operasi euclidean distance untuk algoritma brute force : 120
Jarak antar titik : 6.856

Dengan menggunakan algoritma divide and conquer didapat hasil sebagai berikut :
Titik 1: (82.446, 69.620, 24.186)
Titik 2 : (88.615, 72.572, 23.713)
Waktu : 0.000 ms
Jumlah operasi euclidean distance untuk algoritma divide and conquer : 19
Jarak antar titik : 6.856
Apabila terdapat perbedaan titik, namun jarak euclidean sama hal ini mungkin terjadi apabila ada 2 pasangan titik yang m
emiliki jarak euclidean yang sama
```



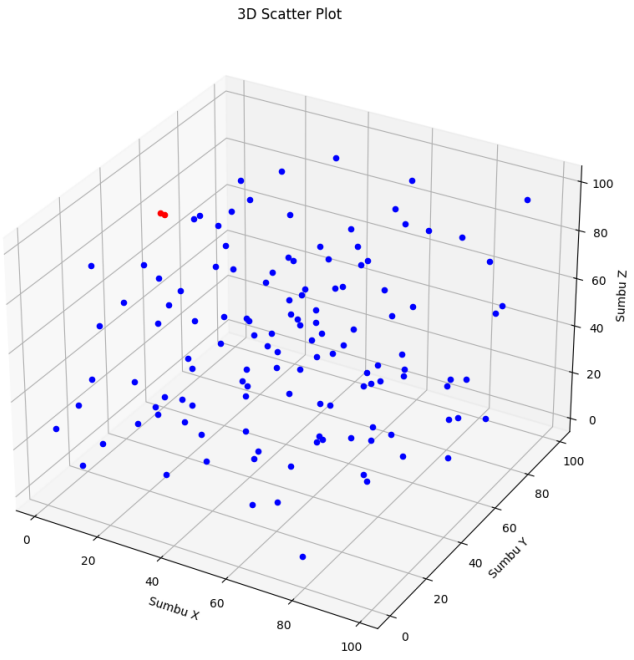
Gambar 4.1 *Test Case* n = 16

```
Masukkan berapa titik yang ingin di generate: 64
Ingin berapa dimensi: 3
Dengan menggunakan algoritma brute force didapat hasil sebagai berikut :
Titik 1 : (1.793, 47.425, 94.365)
Titik 2 : (6.535, 45.452, 92.868)
Waktu : 2.999 ms
Jumlah operasi euclidean distance untuk algoritma brute force : 2016
Jarak antar titik : 5.349

Dengan menggunakan algoritma divide and conquer didapat hasil sebagai berikut :
Titik 1: (1.793, 47.425, 94.365)
Titik 2 : (6.535, 45.452, 92.868)
Waktu : 1.000 ms
Jumlah operasi euclidean distance untuk algoritma divide and conquer : 94
Jarak antar titik : 5.349
Apabila terdapat perbedaan titik, namun jarak euclidean sama hal ini mungkin terjadi apabila ada 2 pasangan titik yang m
emiliki jarak euclidean yang sama
```



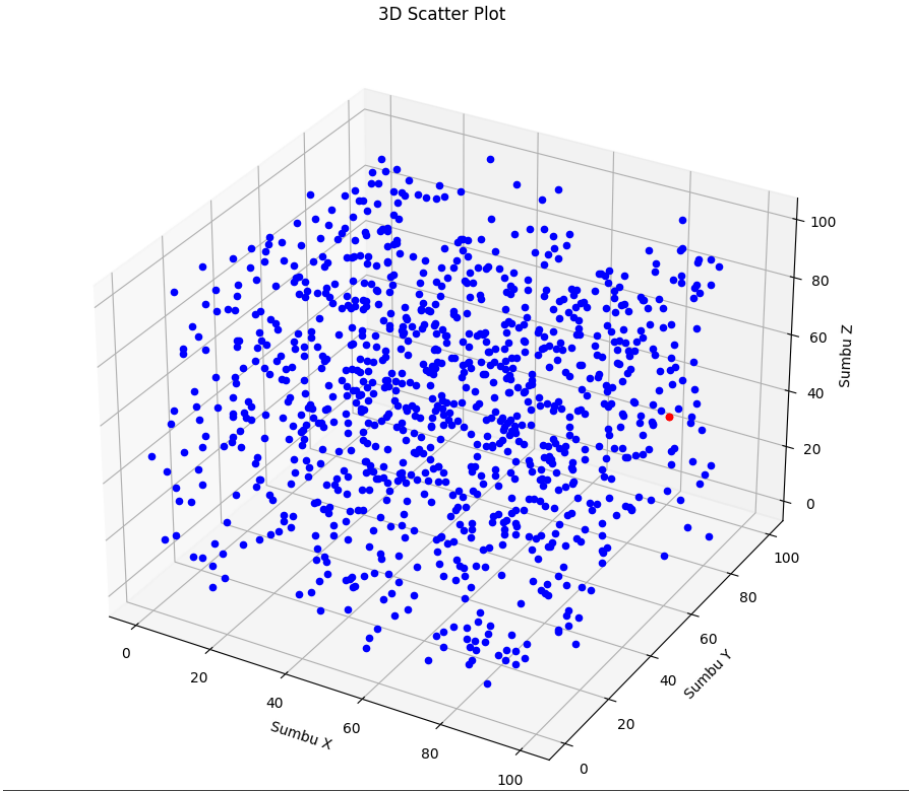
Gambar 4.2 *Test Case* n = 64

```
Masukkan berapa titik yang ingin di generate: 128
Ingin berapa dimensi: 3
Dengan menggunakan algoritma brute force didapat hasil sebagai berikut :
Titik 1 : (46.582, 4.289, 51.847)
Titik 2 : (47.199, 5.590, 52.248)
Waktu : 14.067 ms
Jumlah operasi euclidean distance untuk algoritma brute force : 8128
Jarak antar titik : 1.495

Dengan menggunakan algoritma divide and conquer didapat hasil sebagai berikut :
Titik 1: (46.582, 4.289, 51.847)
Titik 2 : (47.199, 5.590, 52.248)
Waktu : 3.520 ms
Jumlah operasi euclidean distance untuk algoritma divide and conquer : 167
Jarak antar titik : 1.495
Apabila terdapat perbedaan titik, namun jarak euclidean sama hal ini mungkin terjadi apabila ada 2 pasangan titik yang m
emiliki jarak euclidean yang sama
```



Gambar 4.3 *Test Case* n = 128

```
Masukkan berapa titik yang ingin di generate: 1000
Ingin berapa dimensi: 3
Dengan menggunakan algoritma brute force didapat hasil sebagai berikut :
Titik 1 : (65.585, 81.093, 23.879)
Titik 2 : (65.906, 81.050, 23.932)
Waktu : 2020.367 ms
Jumlah operasi euclidean distance untuk algoritma brute force : 499500
Jarak antar titik : 0.329

Dengan menggunakan algoritma divide and conquer didapat hasil sebagai berikut :
Titik 1: (65.585, 81.093, 23.879)
Titik 2 : (65.906, 81.050, 23.932)
Waktu : 419.158 ms
Jumlah operasi euclidean distance untuk algoritma divide and conquer : 1393
Jarak antar titik : 0.329
Apabila terdapat perbedaan titik, namun jarak euclidean sama hal ini mungkin terjadi apabila ada 2 pasangan titik yang m
emiliki jarak euclidean yang sama
```



Gambar 4.4 *Test Case* n = 1000

REFERENSI

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tucil2-Stima-2023.pdf
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

LAMPIRAN

Tautan repository tugas kecil : https://github.com/MarcelRyan/Tucil2_13521054_13521127

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