Tugas Kecil 2 IF2211 Strategi Algoritma Semester II tahun 2022/2023
Pencarian *Closest Pair* Titik dalam 3D dengan Pendekatan *Divide and Conquer*

Disusun Oleh

Farizki Kurniawan	13521082
Akhmad Setiawan	13521164

PROGRAM STUDI TEKNIK INFORMATIKA SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Mencari sepasang titik terdekat (*closest pair*) dengan Algoritma Divide and Conquer sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Pada Tucil 2 kali ini 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 $P_1 = (x_1, y_1, z_1)$ 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.

Masukan program:

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

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, penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya
- Bonus 2, generalisasi program sehingga dapat mencari sepasang titik terdekat untuk sekumpulan vektor di R^N, setiap vektor dinyatakan dalam bentuk $x = (x_1, x_2, \dots, x_n)$

BAB II
LANDASAN TEORI

Divide and Conquer dulunya adalah strategi militer yang dikenal dengan nama *divide ut imperes*, yaitu strategi adu domba untuk memecah belah pihak musuh sehingga lebih rapuh dan mudah ditaklukkan. Dengan konsep yang sama, sekarang strategi tersebut menjadi strategi fundamental di dalam ilmu komputer dengan nama Divide and Conquer.

Seperti namanya, algoritma ini terbagi menjadi beberapa proses eksekusi, yaitu *divide*, yang berarti membagi persoalan menjadi beberapa sub-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya berukuran hampir sama); *conquer*, yakni menyelesaikan masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar); serta proses *combine* yakni menggabungkan solusi masing-masing sub-persoalan sehingga membentuk solusi persoalan semula. Skema umum algoritma *Divide and Conquer* ini dijelaskan seperti *pseudocode* di bawah.

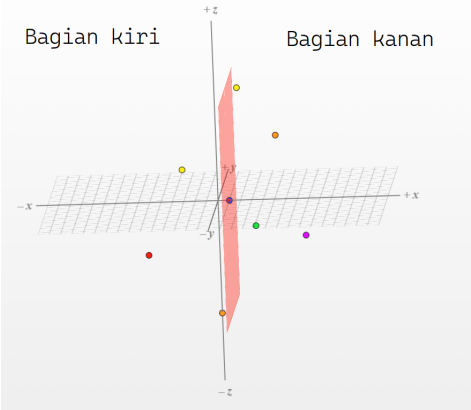
```
procedure DIVIDEandCONQUER(input P : problem, n : integer)
{ Menyelesaikan persoalan P dengan algoritma divide and conquer
  Masukan: masukan persoalan P berukuran n
  Luaran: solusi dari persoalan semula   }
Deklarasi
   r : integer

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
```

Disadur dari: Homepage Rinaldi Munir, Strategi Algoritma
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/stima22-23.htm

Algoritma *Divide and Conquer* dapat digunakan untuk memecahkan berbagai macam persoalan, salah satunya adalah pencarian titik terdekat, baik dalam bidang 1D, 2D, 3D, bahkan dapat digeneralisasi di ruang R^N. 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 = (x_1, y_1, z_1) dan $P_2 = (x_2, y_2, z_2)$ dihitung dengan rumus Euclidean.

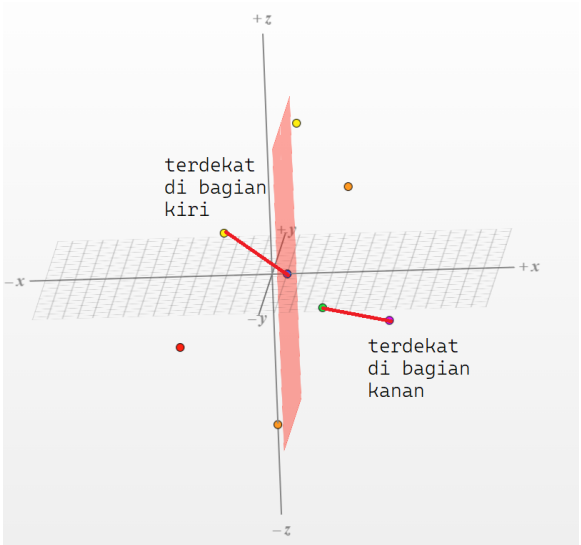
BAB III
IMPLEMENTASI ALGORITMA DIVIDE AND CONQUER

Implementasi fungsi pencarian pasangan titik terdekat di 3 dimensi dilakukan dengan pertama-tama membagi titik-titik yang ada menjadi 2 bagian. Kedua bagian ini akan disebut bagian kanan dan bagian kiri. Pembagian ini dilakukan berdasarkan sumbu x, tepatnya berdasarkan titik pada sumbu x yang membagi titik-titik yang ada menjadi tepat 2 bagian yang kurang-lebih sama banyaknya. Pembagian titik ini merupakan bagian *divide* dari fungsi ini.



Gambar 3.1 Pembagian titik menjadi bagian kanan dan bagian kiri

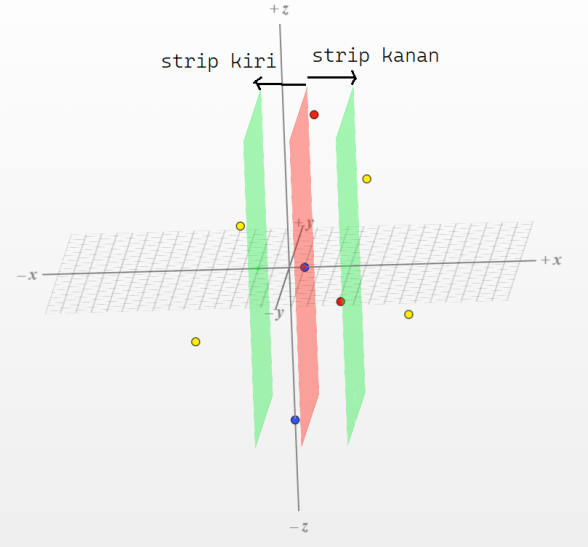
Masing-masing dari kedua bagian ini kemudian akan dicari pasangan titik terdekatnya dengan menggunakan fungsi pencarian pasangan titik terdekat yang sama. Ini adalah bagian *conquer* dari fungsi ini. Terdapat kasus basis berupa saat jumlah titik bernilai 1 dengan hasil keluaran tak hingga, dan saat jumlah titik bernilai 2 dengan hasil keluaran jarak antara kedua titik tersebut. Kita akan mengambil nilai yang paling kecil antara kedua hasil ini dan menamakannya sigma.



Gambar 3.2 Nilai titik terdekat pada masing-masing bagian

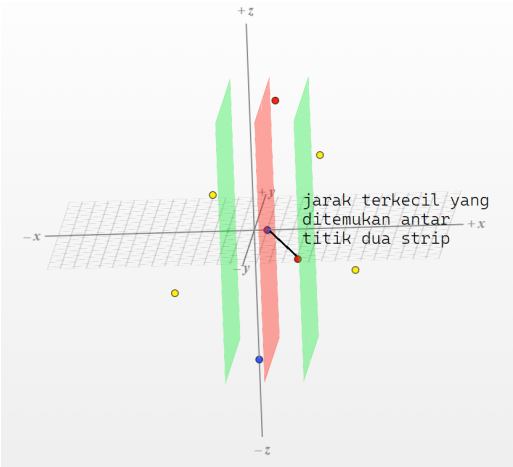
Setelah didapatkan jarak terkecil antara dua titik pada kedua bagian, maka akan dilakukan sebuah ‘penggabungan’ menggunakan algoritma *merge* antara kedua bagian tersebut. Di bagian ini, kita akan mencari pasangan titik dari bagian kanan dan bagian kiri yang memiliki jarak lebih kecil daripada jarak terkecil pada bagian kanan dan bagian kiri.

Algoritma *merge* dimulai dengan membuat strip yang merupakan kumpulan titik yang memiliki jarak **paling jauh** sebesar sigma yang telah didapat sebelumnya. Strip ini juga dibagi menjadi strip yang berisi titik bagian kanan dan strip yang berisi titik bagian kiri.



Gambar 3.3 Strip yang dikonstruksi

Selanjutnya, akan ditinjau titik pada strip kiri dengan urutan pilihan nilai pada suatu sumbu secara menaik. Untuk setiap titik, akan dilakukan traversal pada kumpulan titik strip kanan dengan urutan pada sumbu yang sama secara menaik pula. Traversal berhenti saat beda nilai pada sumbu tersebut antara kedua titik melebihi sigma, yang artinya jarak titik selanjutnya sudah melewati batas yang diinginkan. Apabila ditemukan pasangan titik yang memiliki jarak lebih kecil daripada jarak terkecil sekarang, maka akan dilakukan *update* pada jarak tersebut.



Gambar 3.4 Pasangan titik terdekat yang ditemukan antara 2 strip

Kompleksitas dari algoritma ini dapat ditinjau menggunakan Teorema Master. Dapat kita lihat bahwa suatu fungsi closestPairnD() akan memanggil fungsi closestPairnD() untuk 2 bagian yang berbeda. Hal ini dapat ditunjukkan dengan persamaan sebagai berikut.

$$T(n) = 2T(n/2) + x$$

Kemudian, akan dilakukan sorting pada titik-titik yang berada pada strip, yaitu titik-titik yang berada pada jarak sigma dari bidang yang telah ditentukan. Sorting dilakukan dengan menggunakan *merge sort* dan yang memiliki kompleksitas $O(log\ n)$. Persamaan sebelumnya kemudian dapat dituliskan menjadi:

$$T(n) = 2T(n/2) + O(log\ n) + y$$

Terakhir, dilakukan peninjauan terhadap elemen-elemen pada strip bagian kiri terhadap elemen-elemen pada strip bagian kanan. Meskipun langkah ini terlihat seperti memiliki kompleksitas $O(log\ n^2)$, langkah ini sebenarnya memiliki jumlah operasi sebesar cn , dengan c bergantung pada dimensi, yang berarti kompleksitasnya adalah sebesar $O(n)$. Hal ini disebabkan adanya kondisi penting yaitu **nilai sigma**, dimana dapat dipastikan bahwa jarak terdekat antara dua titik pada suatu bagian ialah sigma, sehingga titik-titik tidak akan bertumpukan, melainkan memiliki batasan tertentu pada radius sigma tersebut. Maka, persamaan kompleksitas untuk algoritma ini ialah

$$T(n) = 2T(n/2) + O(log\ n) + y$$

Dengan Teorema Master, didapatkan kompleksitas algoritma sebesar $O(n\ (log\ n)^2)$.

Untuk kasus dimensi=1, algoritmanya cukup sederhana, yaitu dengan melakukan sorting terhadap titik-titik yang ada, kemudian membandingkan setiap titik dengan titik selanjutnya pada list titik yang telah diurutkan. Kompleksitasnya ialah $O(nlog\ n)$.

Untuk kasus dimensi secara general (dimensi >=2), algoritma yang digunakan sama dengan kasus dimensi 3 yang telah dijelaskan. Namun, mengingat bahwa jumlah operasi pengecekan antarstrip ialah $T(n) = cn$, dimana $T(n)$ memiliki batas atas tertentu. Perlu diketahui bahwa nilai c akan menjadi semakin besar seiring meningkatnya dimensi, sehingga pada dimensi yang sangat besar, kompleksitas algoritma total dapat berubah menjadi $O(n^2)$.

BAB IV
SOURCE CODE PROGRAM

1. pointsGenerator.py

File ini berisi method generateRandomPoints() untuk membangkitkan titik-titik secara acak dengan memanfaatkan *library* random. Titik-titik yang dibangkitkan memiliki rentang antara -10000 hingga 10000 untuk dimensi 1-5, sedangkan untuk 5 ke atas hanya di rentang -1000 hingga 1000. Titik yang dimasukkan sesuai berdasarkan input pengguna, terdapat validasi untuk input pengguna sehingga titik-titik yang akan dibangkitkan minimal 2 titik dan jumlahnya bulat (input n adalah integer) serta validasi input dimensi N (minimal 1 dan integer). Titik-titik yang dibangkitkan kemudian disimpan ke dalam array untuk di-*return* bersama dengan dimensi N.

src > pointsGenerator.py > generateRandomPoints

```python
import random

def generateRandomPoints():
    # validasi input
    while True:
        n = input("How many points to be randomly generated: ") # number of points to generate
        if (n.isdigit() and int(n) > 1):
            n = int(n)
            break
        elif(n.isdigit() == False):
            print("Input must be an integer\n")
        else:
            print("There's should be atleast 2 points to do so!\n")
    while True:
        N = input("Input the dimension: ") # number of dimension (in space R^N)
        if (N.isdigit() and int(N) > 0):
            N = int(N)
            break
        elif(N.isdigit() == False):
            print("Input must be an integer\n")
        else:
            print("The dimension should be atleast 1\n")

    points = []

    for i in range(n):
        point = []
        for j in range(N):
            # random float between 1000000 and 1000000
            # array representing the point, rounded to 3 floating numbers
            if(N<=5):
                point.append(round(random.uniform(-10000, 10000), 3))
            else:
                point.append(round(random.uniform(-1000, 1000), 3))
        points.append(point)
    return points, N
```

Gambar 4.1 Kode Program pointsGenerator.py

2. **DNC.py**

Algoritma *Divide and Conquer* yang digunakan seperti yang telah dijelaskan pada bab sebelumnya. Adapun kode programnya adalah sebagai berikut.

```python
from utilities import *

# Make strip given conditions
def stripMake(leftpoints,rightpoints,mid,sigma,gap=0):
    leftstrip=[]
    rightstrip=[]

    for p in leftpoints:
        if abs(p[gap]-mid)<=sigma:
            leftstrip.append(p)
    for p in rightpoints:
        if abs(p[gap]-mid)<=sigma:
            rightstrip.append(p)

    leftstrip=mergeSort(leftstrip, 1,1)
    rightstrip=mergeSort(rightstrip, 1,1)
    return leftstrip,rightstrip

# divide points to left and right
def divide(points,gap=0):
    N=len(points)

    # sort points
    points=mergeSort(points,1,gap)

    # get midpoint
    midpoint= points[N//2][gap]

    # divide points to 2
    leftpoints=points[:N//2]
    rightpoints=points[N//2:]

    return leftpoints,rightpoints,midpoint

```

```python

# closest pair in 1d
def closestPair1D(points):
    N=len(points)
    points=mergeSort(points,1)
    minDist=inf
    closestPair=None
    n_euclidean_computing=0
    for i in range(N-1):
        distance=euclideanDistance(points[i],points[i+1])
        n_euclidean_computing+=1
        if distance<minDist:
            minDist=distance
            closestPair=[points[i],points[i+1]]
    return[minDist,closestPair,n_euclidean_computing]

# closest pair function
def closestPairnD(points):
    N=len(points)
    dimension=len(points[0])

    # base case
    if dimension==1:
        return closestPair1D(points)
    if N==1:
        return [inf,None,0]
    if N==2:
        return [euclideanDistance(points[0],points[1]),[[points[0],points[1]]],1]

    # if N>2, Divide and Conquer

    # split points to 2
    leftpoints,rightpoints,x0=divide(points)

    # get closest dist in both sets of points, separately
    n_euclidean_computing=0
```

```python
    d1=closestPairnD(leftpoints)
    d2=closestPairnD(rightpoints)

    n_euclidean_computing = d1[2]+d2[2]

    # get closest dist out of the two
    closestPair=None
    d=inf
    if(d1[0]<d2[0]):
        d=d1[0]
        closestPair=d1[1]
    elif(d1[0]>d2[0]):
        d=d2[0]
        closestPair=d2[1]
    else:
        d=d1[0]
        closestPair=d1[1]
        for pairs in d2[1]:
            closestPair.append(pairs)

    #create strip
    leftstrip,rightstrip=stripMake(leftpoints,rightpoints,x0,d)

    # find ans
    minDist=d
    leftNstrip=len(leftstrip)
    rightNstrip=len(rightstrip)

```

```
    for i in range(leftNstrip):
        for j in range(rightNstrip):
            # sparsity condition,
            # don't care if y distance exceed d
            if rightstrip[j][1]-leftstrip[i][1]<-d:
                continue
            if rightstrip[j][1]-leftstrip[i][1]>d:
                break
            temp=euclideanDistance(leftstrip[i],rightstrip[j])
            n_euclidean_computing+=1
            if temp<minDist:
                minDist=temp
                closestPair=[[leftstrip[i],rightstrip[j]]]
            elif temp==minDist:
                closestPair.append([leftstrip[i],rightstrip[j]])

    # done, return minimum distance
    return [minDist,closestPair,n_euclidean_computing]
```

Gambar 4.2 Kode Program DNC.py

3. **bruteForce.py**

   Sebagai pembanding, dibuat solusi untuk mencari pasangan titik terdekat dengan metode *Brute Force* di *file* ini. Terdapat dua method, yakni bruteForceClosestPair3D(points) untuk mencari solusi pada dimensi 3 dan bruteForceClosestPairnD(points) sebagai generalisasi pencarian solusi pada dimensi N. Kedua method tersebut sama-sama me-return jarak minimum pasangan titik, pasangan titik terdekat, dan banyaknya operasi euclidean yang terjadi.

src > bruteForce.py > bruteForceClosestPairnD

```python
# Brute Force Algorithm for Finding Closest Pair of Points
def bruteForceClosestPair3D(points):
    n_euclidean_computing = 0
    n = len(points)
    min_distance = inf
    closest_pair = []
    for i in range(n):
        for j in range(i+1, n):
            distance = sqrt((points[i][0]-points[j][0])**2 +
                            (points[i][1]-points[j][1])**2 +
                            (points[i][2]-points[j][2])**2)
            n_euclidean_computing += 1
            if (distance < min_distance):
                min_distance = distance
                closest_pair = []
                closest_pair.append([points[i], points[j]])
            elif (distance == min_distance):
                closest_pair.append([points[i], points[j]])
    return [min_distance, closest_pair, n_euclidean_computing]

def bruteForceClosestPairnD(points):
    n_euclidean_computing = 0
    n = len(points)
    min_distance = inf
    closest_pair = None
    for i in range(n):
        for j in range(i+1, n):
            distance = sqrt(sum([(points[i][k]-points[j][k])**2 for k in range(len(points[i]))]))
            n_euclidean_computing += 1
            if (distance < min_distance):
                min_distance = distance
                closest_pair = []
                closest_pair.append([points[i], points[j]])
            elif (distance == min_distance):
                closest_pair.append([points[i], points[j]])
    return [min_distance, closest_pair, n_euclidean_computing]
```

Gambar 4.3 Kode Program bruteForce.py

4. **utilities.py**

File ini berisi method-method yang diimplementasikan untuk membantu jalannya program. Terdapat method mergeSort dan merge sebagai implementasi dari algoritma sorting titik yang telah dibangkitkan tadi. Sorting dengan merge sort ini diimplementasikan dengan algoritma *Divide and Conquer* pula, yakni dengan membagi-bagi titik (dalam array) lalu di-*solve* dan di-*combine* untuk kemudian menghasilkan urutan titik yang diinginkan. Sorting yang diimplementasikan dapat berupa urutan menaik (*ascending*) dan menurun (*descending*) sesuai dengan parameter yang diinput.

src > utilities.py > merge

```python
from math import *
import matplotlib.pyplot as plt
import numpy as np

# Merge Sort Algorithm Using Divide and Conquer Approach for Array of Points
# in default, sorting would be ascending
def mergeSort(arr, asc = 1, el=0):
    if len(arr) <= 1:
        return arr

    # Divide
    mid = len(arr) // 2
    left_arr = arr[:mid]
    right_arr = arr[mid:]

    left_arr = mergeSort(left_arr, asc, el)
    right_arr = mergeSort(right_arr, asc, el)

    # conquer
    return merge(left_arr, right_arr, asc, el)
```

```python
# conquer implementation
def merge(left_arr, right_arr, asc,el):
    result = []
    left_idx = 0
    right_idx = 0

    while left_idx < len(left_arr) and right_idx < len(right_arr):

        # if ascending is True or 1
        if (asc == 1):
            if left_arr[left_idx][el] <= right_arr[right_idx][el]:
                result.append(left_arr[left_idx])
                left_idx += 1
            else:
                result.append(right_arr[right_idx])
                right_idx += 1

        # if ascending is False or any other than 1, hence descending
        else:
            if left_arr[left_idx][el] >= right_arr[right_idx][el]:
                result.append(left_arr[left_idx])
                left_idx += 1
            else:
                result.append(right_arr[right_idx])
                right_idx += 1

    if left_idx < len(left_arr):
        result.extend(left_arr[left_idx:])
    if right_idx < len(right_arr):
        result.extend(right_arr[right_idx:])

    return result
```

Terdapat pula rumus euclidean untuk mencari jarak antara dua titik.

```
56   # find distance between 2 points
57   def euclideanDistance(a,b):
58       dimension=len(a)
59       temp=0
60       for i in range (dimension):
61           temp += (a[i]-b[i])*(a[i]-b[i])
62       return sqrt(temp)
```

File ini juga menyimpan method untuk memvisualisasikan titik-titik pada bidang 3D. Pasangan titik yang terdekat akan diberi warna merah. Visualisasi hanya untuk titik dengan dimensi 1D, 2D, dan 3D, sedangkan untuk dimensi di atas 3 tidak dapat divisualisasikan. Visualisasi dibantu dengan *library* matplotlib.

```
63
64   # Create a 3D scatter plot of the points
65   def visualize3D(points, closest_pair_list):
66       fig = plt.figure()
67       ax = plt.axes(projection='3d')
68       ax.set_xlabel('X')
69       ax.set_ylabel('Y')
70       ax.set_zlabel('Z')
71
72       for i in range(len(points)):
73           flag = False
74           for j in range(len(closest_pair_list)):
75               if points[i] in closest_pair_list[j]:
76                   flag = True
77           if not flag:
78               ax.scatter(points[i][0], points[i][1], points[i][2], c='b')
79           else:
80               ax.scatter(points[i][0], points[i][1], points[i][2], c='r')
81
82       for i in range(len(closest_pair_list)):
83           x = [closest_pair_list[i][0][0], closest_pair_list[i][1][0]]
84           y = [closest_pair_list[i][0][1], closest_pair_list[i][1][1]]
85           z = [closest_pair_list[i][0][2], closest_pair_list[i][1][2]]
86           ax.plot(x, y, z, c='r')
87       plt.show()
```

```python
def visualize2D(points, closest_pair_list):
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    for i in range(len(points)):
        flag = False
        for j in range(len(closest_pair_list)):
            if points[i] in closest_pair_list[j]:
                flag = True
        if not flag:
            ax.scatter(points[i][0], points[i][1], c='b')
        else:
            ax.scatter(points[i][0], points[i][1], c='r')

    for i in range(len(closest_pair_list)):
        x = [closest_pair_list[i][0][0], closest_pair_list[i][1][0]]
        y = [closest_pair_list[i][0][1], closest_pair_list[i][1][1]]
        ax.plot(x, y, c='r')
    plt.show()
```

```python
def visualize1D(points, closest_pair_list):
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    for i in range(len(points)):
        flag = False
        for j in range(len(closest_pair_list)):
            if points[i] in closest_pair_list[j]:
                flag = True
        if not flag:
            ax.scatter(points[i][0], 0, 0, c='b')
        else:
            ax.scatter(points[i][0], 0, 0, c='r')

    for i in range(len(closest_pair_list)):
        x = [closest_pair_list[i][0][0], closest_pair_list[i][1][0]]
        ax.plot(x, [0, 0], [0, 0], c='r')
    plt.show()
```

Gambar 4.4 Kode Program utilities.py

5. **main.py**

Seluruh file-file di atas kemudian di-import ke dalam main.py, file untuk menjalankan program utama. Program utama berisi output-output yang akan ditampilkan pada terminal serta validasi untuk menanyakan apakah ingin melihat visualisasi titik-titik pada bidang 3D.

src > main.py > ...

```python
from DNC import *
from bruteForce import *
from pointsGenerator import *
from utilities import *
import time

# Generate Points
points, dimension = generateRandomPoints()

# Finding Closest Pair with Divide n Conquer Approach
start = time.time()
result = closestPairnD(points)
end = time.time()

print("\nWith Divide and Conquer Approach:")
print("the closest pair:", end=" ")
for i in result[1]:
    print(i, end=' ')
print("\nwith the minimum distance:", round(result[0], 3))
print("the euclidean operation executed by:", result[2], "times")
print("and execution time within:", end=" ")
print("{0:.10f}".format(end-start), "seconds\n")

# Finding Closest Pair with Brute Force Approach
start = time.time()
result = bruteForceClosestPairnD(points)
end = time.time()

print("\nWith Brute Force Approach:")
print("the closest pair:", end=" ")
for i in result[1]:
    print(i, end=' ')
print("\nwith the minimum distance:", round(result[0], 3))
print("the euclidean operation executed by:", result[2], "times")
print("and execution time within:", end=" ")
print("{0:.10f}".format(end-start), "seconds")
```

```python
# Asking for visualization
if (dimension == 3):
    while True:
        visualize = input("Do you want to visualize the points? (y/n): ")
        if (visualize == 'y'):
            visualize3D(points, result[1])
            break
        elif (visualize == 'n'):
            break
        else:
            print("Invalid input!")

elif (dimension == 2):
    while True:
        visualize = input("Do you want to visualize the points? (y/n): ")
        if (visualize == 'y'):
            visualize2D(points, result[1])
            break
        elif (visualize == 'n'):
            break
        else:
            print("Invalid input!")

elif (dimension == 1):
    while True:
        visualize = input("Do you want to visualize the points? (y/n): ")
        if (visualize == 'y'):
            visualize1D(points, result[1])
            break
        elif (visualize == 'n'):
            break
        else:
            print("Invalid input!")
```

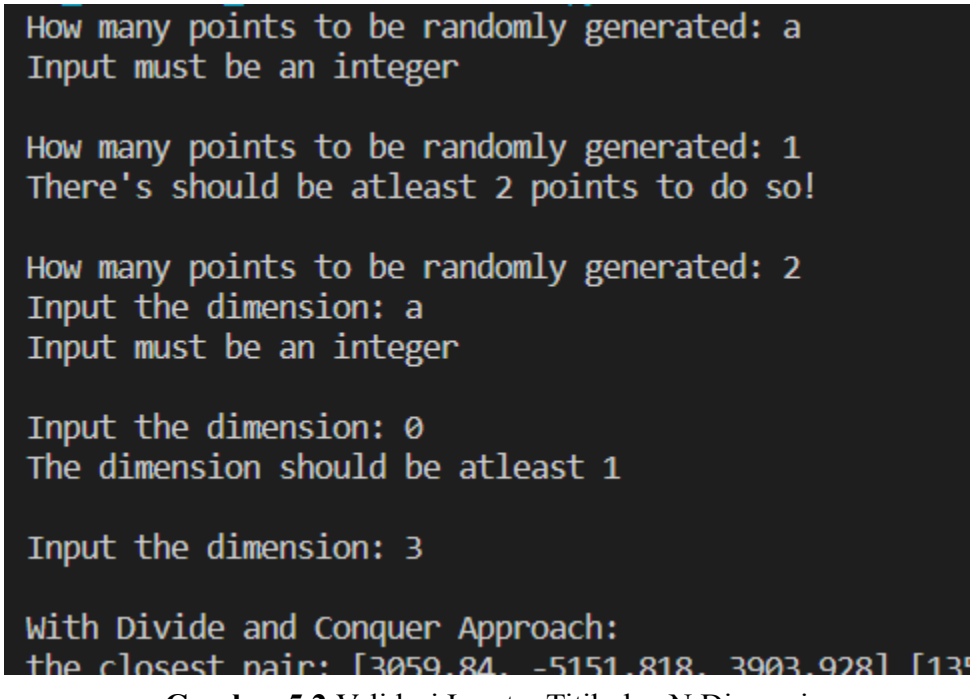
Gambar 4.5 Kode Program Utama main.py

BAB V
HASIL PENGUJIAN

Program ini diuji pada Lenovo Ideapad Gaming 3 15ACH6 dengan sistem operasi Windows.

Gambar 5.1 Spesifikasi Komputer Penguji

Uji validasi input pada input banyak titik n dan banyak dimensi N.



Gambar 5.2 Validasi Input n Titik dan N Dimensi

Uji validasi input untuk menampilkan visualisasi

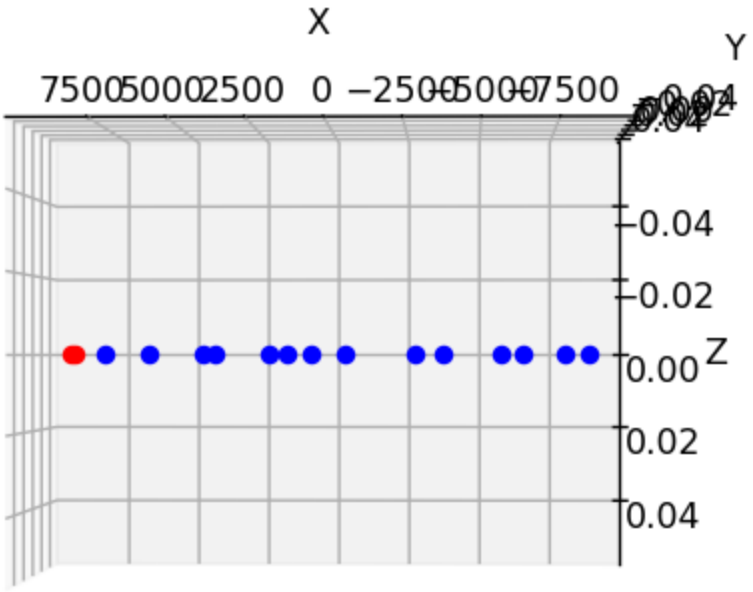
```
Do you want to visualize the points? (y/n): s
Invalid input!
Do you want to visualize the points? (y/n): 3
Invalid input!
Do you want to visualize the points? (y/n): 1
Invalid input!
Do you want to visualize the points? (y/n): n
PS C:\Users\Lenovo\Documents\Semester 4\Tucil2 13
```

Gambar 5.2 Validasi Input Visualisasi Titik

1. **Pengujian untuk n = 16**
   a. Dimensi 1

```
How many points to be randomly generated: 16
Input the dimension: 1

With Divide and Conquer Approach:
the closest pair: [8660.543] [8809.201]
with the minimum distance: 148.658
the euclidean operation executed by: 15 times
and execution time within: 0.0000000000 seconds


With Brute Force Approach:
the closest pair: [[8660.543], [8809.201]]
with the minimum distance: 148.658
the euclidean operation executed by: 120 times
and execution time within: 0.0000000000 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.3 Hasil Pengujian 16 titik 1 Dimensi

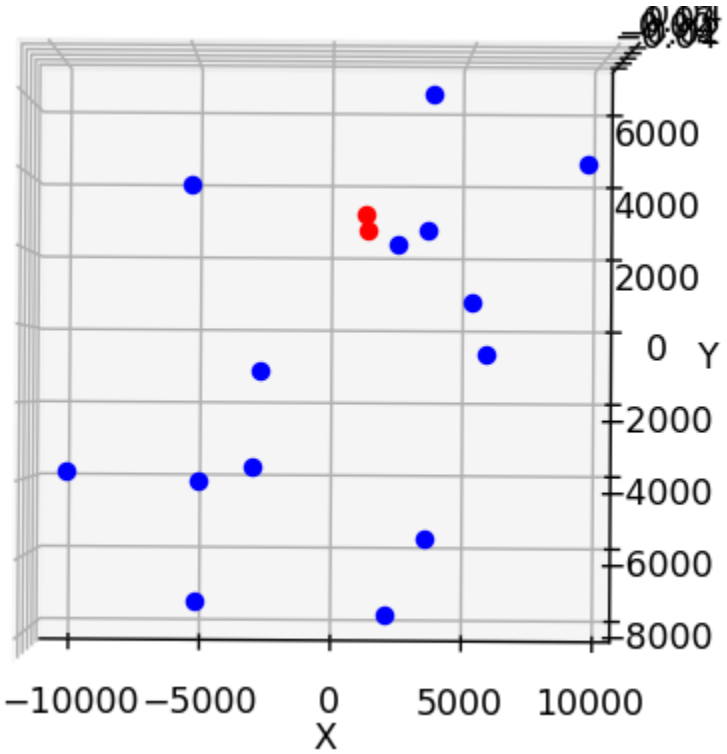


Gambar 5.4 Visualisasi 16 Titik 1 Dimensi

b. Dimensi 2

```
How many points to be randomly generated: 16
Input the dimension: 2

With Divide and Conquer Approach:
the closest pair: [1199.178, 3013.744] [1291.356, 2588.533]
with the minimum distance: 435.088
the euclidean operation executed by: 14 times
and execution time within: 0.0000000000 seconds


With Brute Force Approach:
the closest pair: [[1199.178, 3013.744], [1291.356, 2588.533]]
with the minimum distance: 435.088
the euclidean operation executed by: 120 times
and execution time within: 0.0000000000 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.5 Hasil Pengujian 16 Titik 2 Dimensi

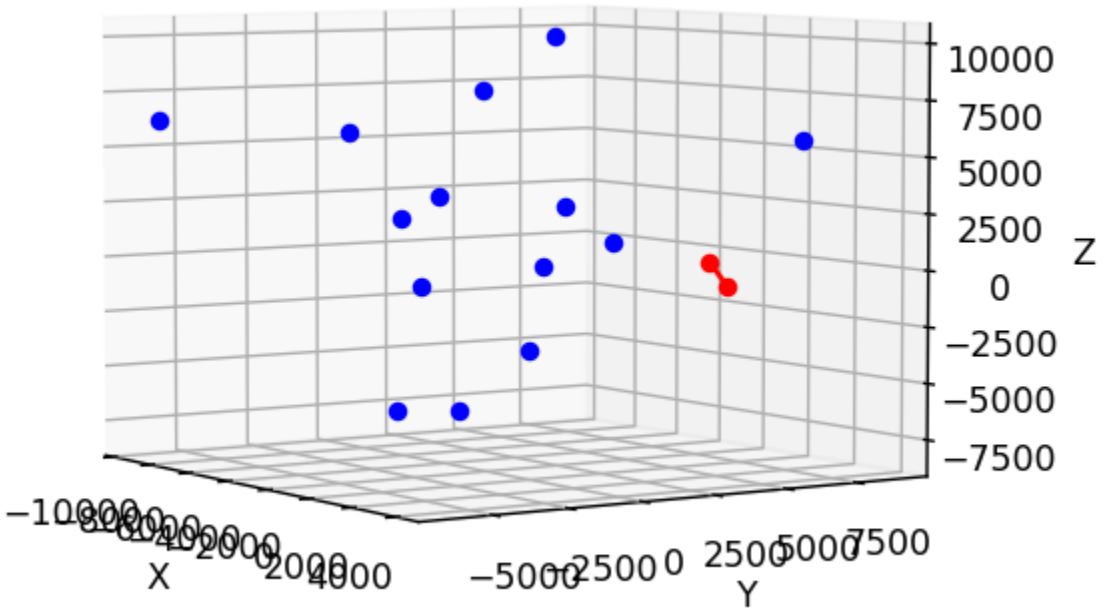


Gambar 5.6 Visualisasi 16 Titik 2 Dimensi

c. Dimensi 3

```
How many points to be randomly generated: 16
Input the dimension: 3

With Divide and Conquer Approach:
the closest pair: [2156.712, 4622.294, 438.045] [4080.549, 3690.64, -378.557]
with the minimum distance: 2288.224
the euclidean operation executed by: 22 times
and execution time within: 0.0000000000 seconds


With Brute Force Approach:
the closest pair: [[2156.712, 4622.294, 438.045], [4080.549, 3690.64, -378.557]]
with the minimum distance: 2288.224
the euclidean operation executed by: 120 times
and execution time within: 0.0000000000 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.7 Hasil Pengujian 16 Titik 3 Dimensi



Gambar 5.8 Visualisasi 16 Titik 3 Dimensi

d. Dimensi 6

```
How many points to be randomly generated: 16
Input the dimension: 6

With Divide and Conquer Approach:
the closest pair: [-179.095, -553.896, 131.496, 200.75, 495.659, 245.408] [-125.445, -767.257, -495.413, -34.616, 372.047, 96.99]
with the minimum distance: 730.836
the euclidean operation executed by: 71 times
and execution time within: 0.0000000000 seconds


With Brute Force Approach:
the closest pair: [[-179.095, -553.896, 131.496, 200.75, 495.659, 245.408], [-125.445, -767.257, -495.413, -34.616, 372.047, 96.99]]
with the minimum distance: 730.836
the euclidean operation executed by: 120 times
and execution time within: 0.0000000000 seconds
PS C:\Users\Lenovo\Documents\Semester 4\Tucil2_13521082_13521164>
```

Gambar 5.9 Hasil Pengujian 16 Titik 6 Dimensi

e. Dimensi 11

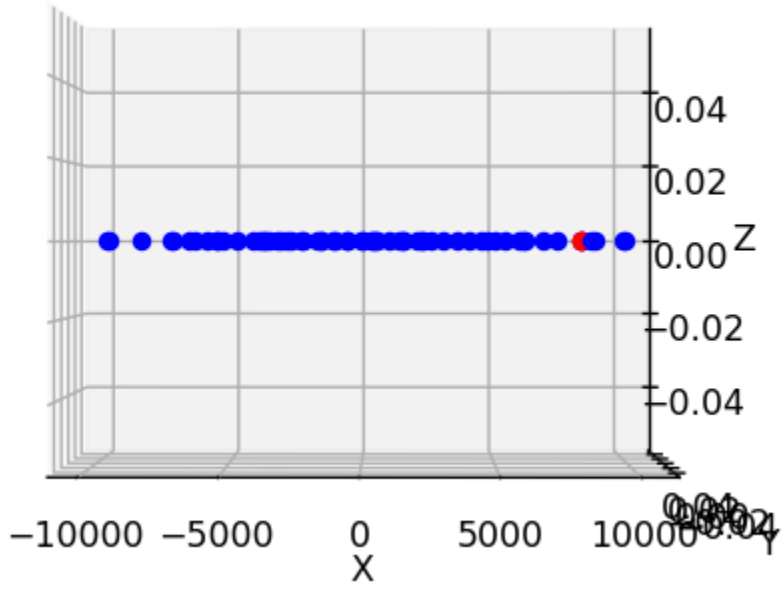
```
How many points to be randomly generated: 16
Input the dimension: 11

With Divide and Conquer Approach:
the closest pair: [460.711, -443.314, -376.129, 368.646, 169.388, -948.757, -497.685, -848.787, -511.245, 648.291, 999.
968] [548.088, 245.27, 176.467, -0.378, 429.252, -595.931, -91.499, -799.071, -940.885, 2.318, 765.385]
with the minimum distance: 1392.731
the euclidean operation executed by: 115 times
and execution time within: 0.0004005432 seconds


With Brute Force Approach:
the closest pair: [[460.711, -443.314, -376.129, 368.646, 169.388, -948.757, -497.685, -848.787, -511.245, 648.291, 999
.968], [548.088, 245.27, 176.467, -0.378, 429.252, -595.931, -91.499, -799.071, -940.885, 2.318, 765.385]]
with the minimum distance: 1392.731
the euclidean operation executed by: 120 times
and execution time within: 0.0010142326 seconds
PS C:\Users\Lenovo\Documents\Semester 4\Tucil2_13521082_13521164>
```

Gambar 5.10 Hasil Pengujian 16 Titik 11 Dimensi

2. **Pengujian untuk n = 64**
    a. Dimensi 1

```
How many points to be randomly generated: 64
Input the dimension: 1

With Divide and Conquer Approach:
the closest pair: [8284.963] [8286.591]
with the minimum distance: 1.628
the euclidean operation executed by: 63 times
and execution time within: 0.0000000000 seconds


With Brute Force Approach:
the closest pair: [[8284.963], [8286.591]]
with the minimum distance: 1.628
the euclidean operation executed by: 2016 times
and execution time within: 0.0000000000 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.11 Hasil Pengujian 64 Titik 1 DImensi

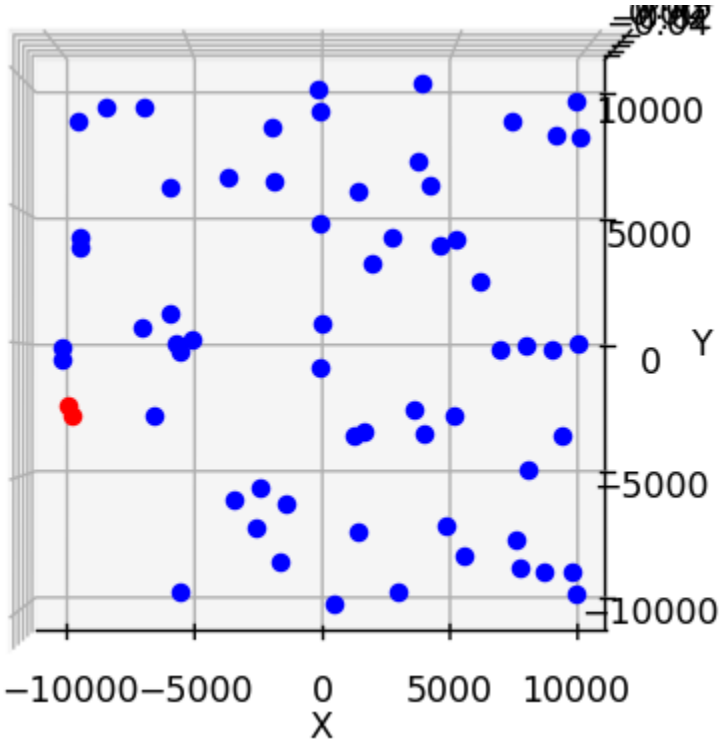


Gambar 5.12 Visualisasi 64 Titik 1 Dimensi

b. Dimensi 2

```
How many points to be randomly generated: 64
Input the dimension: 2

With Divide and Conquer Approach:
the closest pair: [-9513.203, -2372.268] [-9408.296, -2695.919]
with the minimum distance: 340.229
the euclidean operation executed by: 78 times
and execution time within: 0.0010187626 seconds


With Brute Force Approach:
the closest pair: [[-9513.203, -2372.268], [-9408.296, -2695.919]]
with the minimum distance: 340.229
the euclidean operation executed by: 2016 times
and execution time within: 0.0014994144 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.13 Hasil Pengujian 64 Titik 2 Dimensi

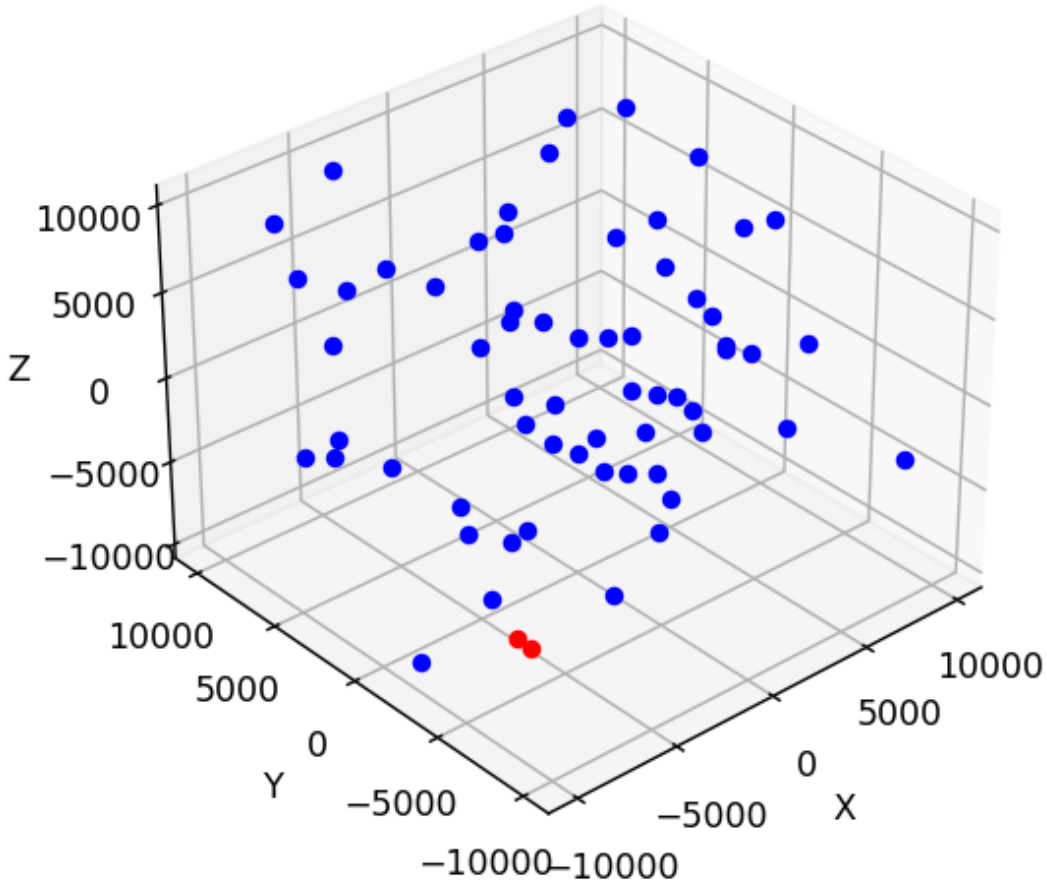


Gambar 5.14 Visualisasi 64 Titik 2 Dimensi

c. Dimensi 3

```
How many points to be randomly generated: 64
Input the dimension: 3

With Divide and Conquer Approach:
the closest pair: [-7412.628, -4999.492, -7366.006] [-7385.705, -5700.944, -7526.119]
with the minimum distance: 719.997
the euclidean operation executed by: 131 times
and execution time within: 0.0007612705 seconds


With Brute Force Approach:
the closest pair: [[-7412.628, -4999.492, -7366.006], [-7385.705, -5700.944, -7526.119]]
with the minimum distance: 719.997
the euclidean operation executed by: 2016 times
and execution time within: 0.0030698776 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.15 Hasil Pengujian 64 Titik 3 Dimensi



Gambar 5.16 Visualisasi 64 Titik 3 Dimensi

d. Dimensi 4

```
How many points to be randomly generated: 64
Input the dimension: 4

With Divide and Conquer Approach:
the closest pair: [8477.864, 2005.747, -9164.643, -7560.23] [9159.669, -337.5, -9284.783, -7872.152]
with the minimum distance: 2463.208
the euclidean operation executed by: 267 times
and execution time within: 0.0010309219 seconds


With Brute Force Approach:
the closest pair: [[8477.864, 2005.747, -9164.643, -7560.23], [9159.669, -337.5, -9284.783, -7872.152]]
with the minimum distance: 2463.208
the euclidean operation executed by: 2016 times
and execution time within: 0.0036816597 seconds
```

Gambar 5.17 Hasil Pengujian 64 Titik 4 Dimensi

e. Dimensi 8

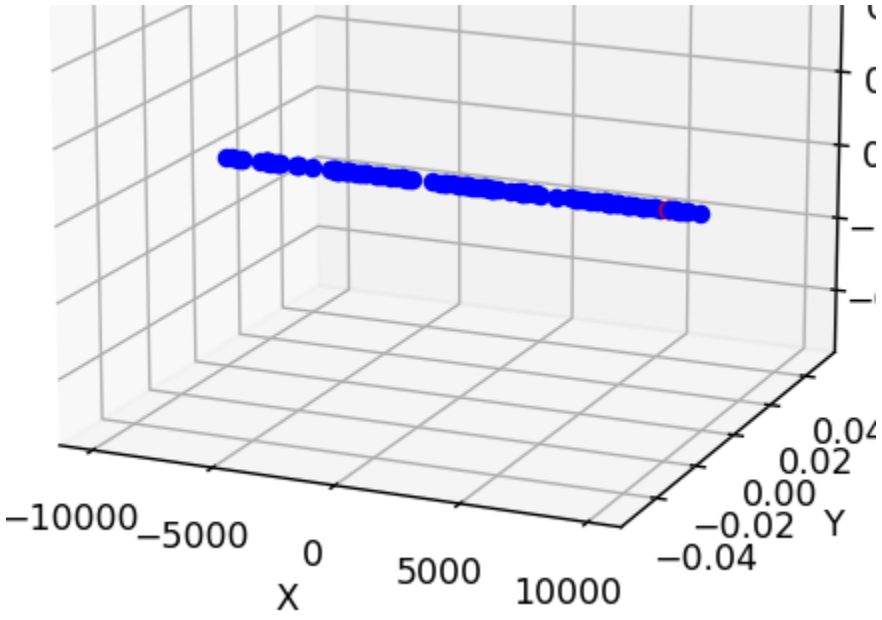
```
How many points to be randomly generated: 64
Input the dimension: 8

With Divide and Conquer Approach:
the closest pair: [-613.161, 341.231, 329.336, -776.526, -504.454, -997.398, 107.144, 66.466] [-490.302, 672.554, -107.
911, -923.776, -507.658, -899.246, 204.947, -344.934]
with the minimum distance: 725.394
the euclidean operation executed by: 1086 times
and execution time within: 0.0030829906 seconds


With Brute Force Approach:
the closest pair: [[-613.161, 341.231, 329.336, -776.526, -504.454, -997.398, 107.144, 66.466], [-490.302, 672.554, -10
7.911, -923.776, -507.658, -899.246, 204.947, -344.934]]
with the minimum distance: 725.394
the euclidean operation executed by: 2016 times
and execution time within: 0.0036931038 seconds
PS C:\Users\Lenovo\Documents\Semester 4\Tucil2_13521082_13521164>
```

Gambar 5.18 Hasil Pengujian 64 Titik 8 Dimensi

3. **Pengujian untuk n = 128**
    a. Dimensi 1

```
How many points to be randomly generated: 128
Input the dimension: 1

With Divide and Conquer Approach:
the closest pair: [8570.273] [8570.808]
with the minimum distance: 0.535
the euclidean operation executed by: 127 times
and execution time within: 0.0023717880 seconds


With Brute Force Approach:
the closest pair: [[8570.808], [8570.273]]
with the minimum distance: 0.535
the euclidean operation executed by: 8128 times
and execution time within: 0.0066678524 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.19 Hasil Pengujian 128 Titik 1 Dimensi

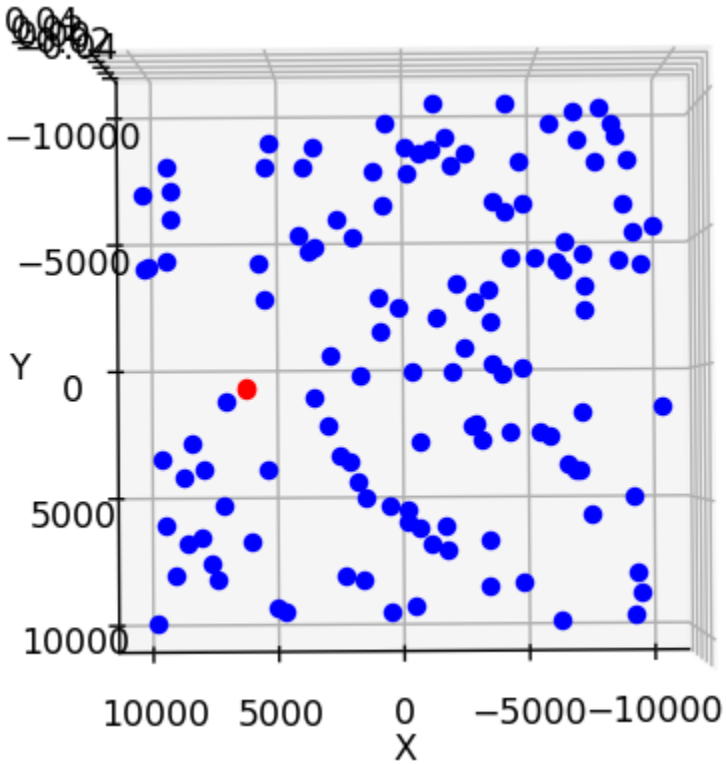


Gambar 5.20 Visualisasi 128 Titik 1 Dimensi

b. Dimensi 2

```
How many points to be randomly generated: 128
Input the dimension: 2

With Divide and Conquer Approach:
the closest pair: [5986.28, 719.633] [6014.905, 801.245]
with the minimum distance: 86.486
the euclidean operation executed by: 122 times
and execution time within: 0.0017552376 seconds


With Brute Force Approach:
the closest pair: [[5986.28, 719.633], [6014.905, 801.245]]
with the minimum distance: 86.486
the euclidean operation executed by: 8128 times
and execution time within: 0.0062370300 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.21 Hasil Pengujian 128 Titik 2 Dimensi

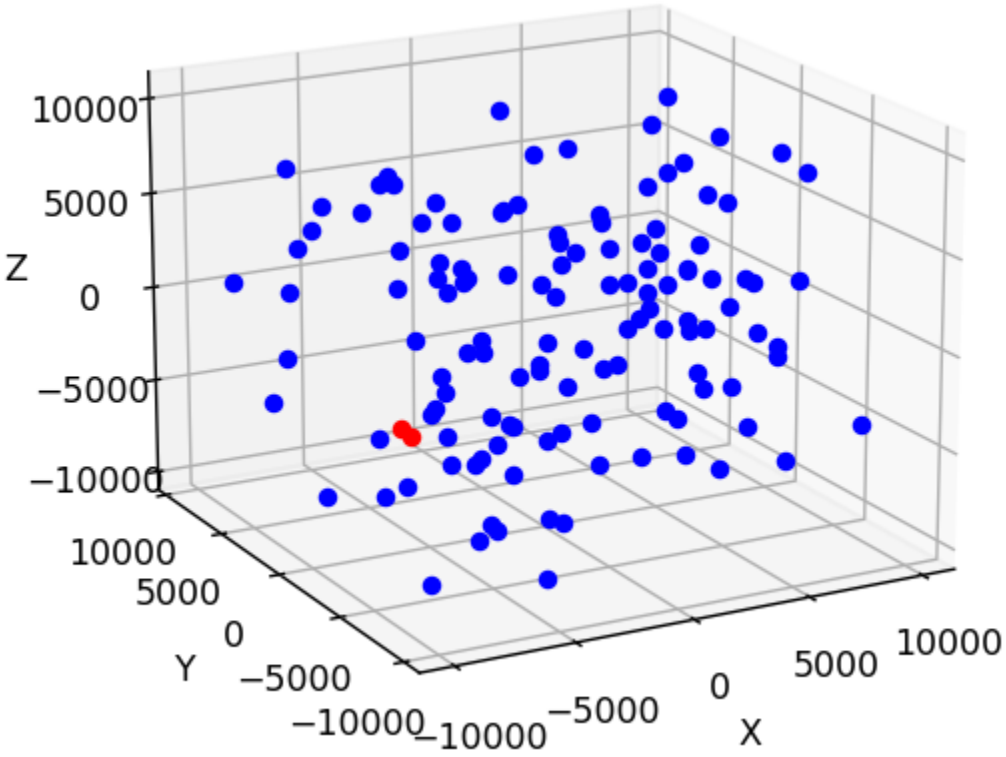


Gambar 5.22 Visualisasi 128 Titik 2 Dimensi

c. Dimensi 3

```
How many points to be randomly generated: 128
Input the dimension: 3

With Divide and Conquer Approach:
the closest pair: [-7766.932, -3041.716, -2950.991] [-7457.232, -3221.69, -3312.096]
with the minimum distance: 508.627
the euclidean operation executed by: 292 times
and execution time within: 0.0036008358 seconds


With Brute Force Approach:
the closest pair: [[-7766.932, -3041.716, -2950.991], [-7457.232, -3221.69, -3312.096]]
with the minimum distance: 508.627
the euclidean operation executed by: 8128 times
and execution time within: 0.0072960854 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.23 Hasil Pengujian 128 Titik 3 Dimensi



Gambar 5.24 Visualisasi 128 Titik 3 Dimensi

d. Dimensi 7

```
How many points to be randomly generated: 128
Input the dimension: 7

With Divide and Conquer Approach:
the closest pair: [-195.35, -361.88, -426.263, 411.746, -888.536, -87.785, 26.197] [99.091, -216.231, -350.145,
139.128, -953.061, -161.085, 5.266]
with the minimum distance: 444.97
the euclidean operation executed by: 2582 times
and execution time within: 0.0064949989 seconds


With Brute Force Approach:
the closest pair: [[-195.35, -361.88, -426.263, 411.746, -888.536, -87.785, 26.197], [99.091, -216.231, -350.145
, 139.128, -953.061, -161.085, 5.266]]
with the minimum distance: 444.97
the euclidean operation executed by: 8128 times
and execution time within: 0.0124785900 seconds
PS C:\Users\Lenovo\Documents\Semester 4\Tucil2 13521082 13521164>
```

Gambar 5.25 Hasil Pengujian 128 Titik 7 Dimensi

e. Dimensi 9

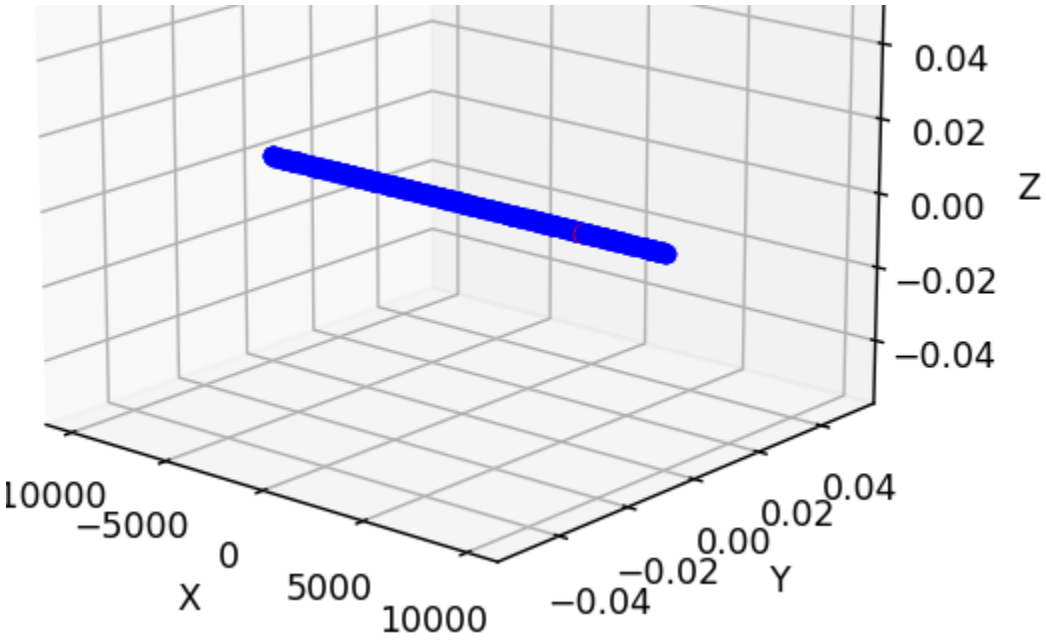
```
How many points to be randomly generated: 128
Input the dimension: 9

With Divide and Conquer Approach:
the closest pair: [-465.321, -46.921, 486.197, 704.72, 629.055, 658.655, -509.216, -285.991, 150.136] [-130.405,
135.299, 640.043, 899.165, 763.007, 547.829, -116.055, -551.432, 28.472]
with the minimum distance: 690.588
the euclidean operation executed by: 3969 times
and execution time within: 0.0067141056 seconds


With Brute Force Approach:
the closest pair: [[-465.321, -46.921, 486.197, 704.72, 629.055, 658.655, -509.216, -285.991, 150.136], [-130.405
, 135.299, 640.043, 899.165, 763.007, 547.829, -116.055, -551.432, 28.472]]
with the minimum distance: 690.588
the euclidean operation executed by: 8128 times
and execution time within: 0.0161616802 seconds
PS C:\Users\Lenovo\Documents\Semester 4\Tucil2 13521082 13521164>
```

Gambar 5.26 Hasil Pengujian 128 Titik 9 Dimensi

4. **Pengujian untuk n = 1000**
    a. Dimensi 1

```
How many points to be randomly generated: 1000
Input the dimension: 1

With Divide and Conquer Approach:
the closest pair: [5993.203] [5993.208]
with the minimum distance: 0.005
the euclidean operation executed by: 999 times
and execution time within: 0.0030078888 seconds


With Brute Force Approach:
the closest pair: [[5993.203], [5993.208]]
with the minimum distance: 0.005
the euclidean operation executed by: 499500 times
and execution time within: 0.3001470566 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.27 Hasil Pengujian 1000 Titik 1 Dimensi

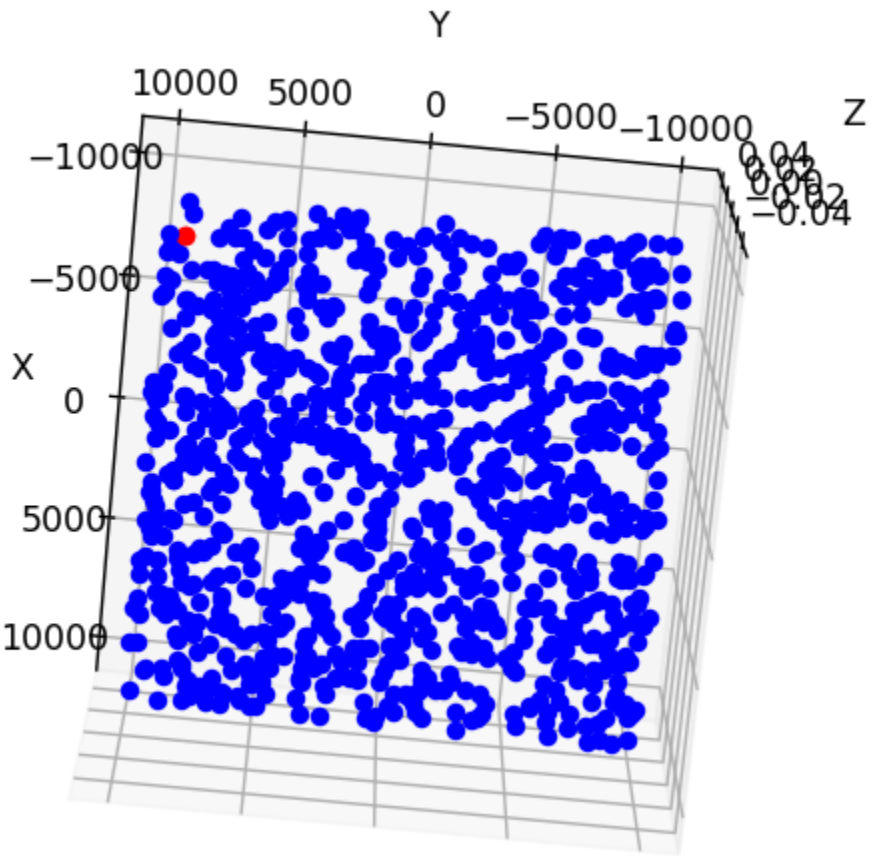


Gambar 5.28 Visualisasi 1000 Titik 1 Dimensi

b. Dimensi 2

```
How many points to be randomly generated: 1000
Input the dimension: 2

With Divide and Conquer Approach:
the closest pair: [-8589.78, 9175.812] [-8583.037, 9182.411]
with the minimum distance: 9.435
the euclidean operation executed by: 1036 times
and execution time within: 0.0146613121 seconds


With Brute Force Approach:
the closest pair: [[-8589.78, 9175.812], [-8583.037, 9182.411]]
with the minimum distance: 9.435
the euclidean operation executed by: 499500 times
and execution time within: 0.3690743446 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.29 Hasil Pengujian 1000 Titik 2 Dimensi

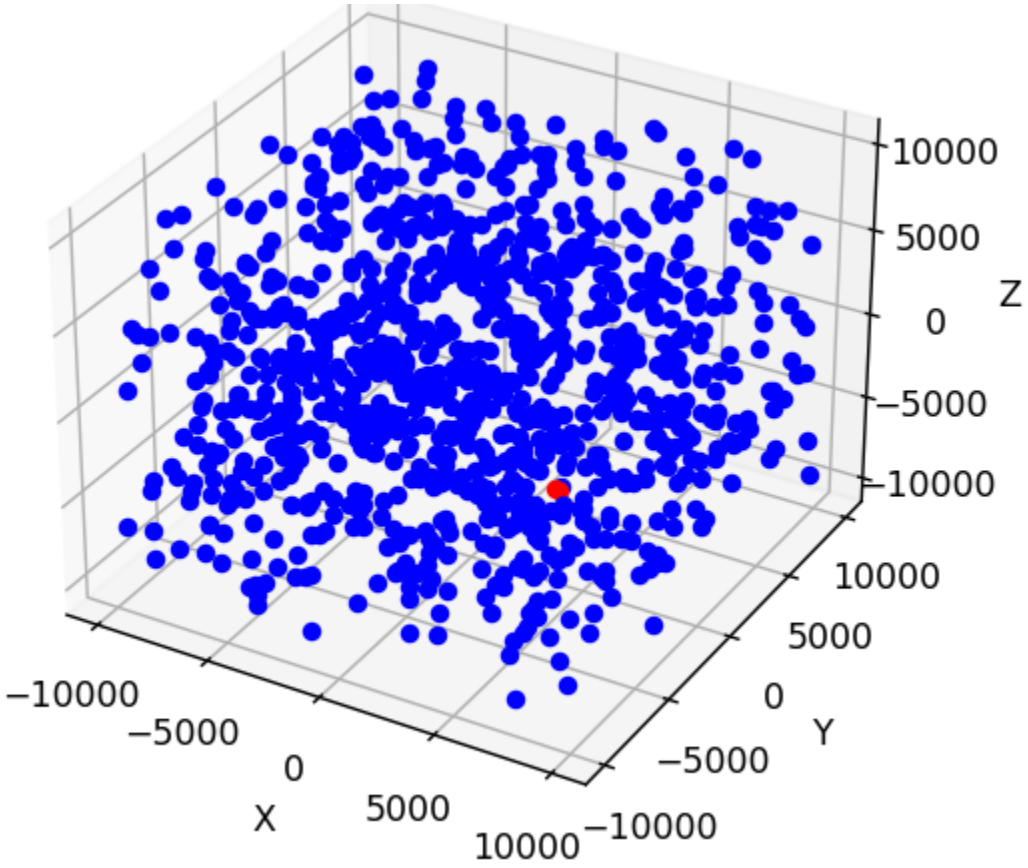


Gambar 5.30 Visualisasi 1000 Titik 2 Dimensi

c. Dimensi 3

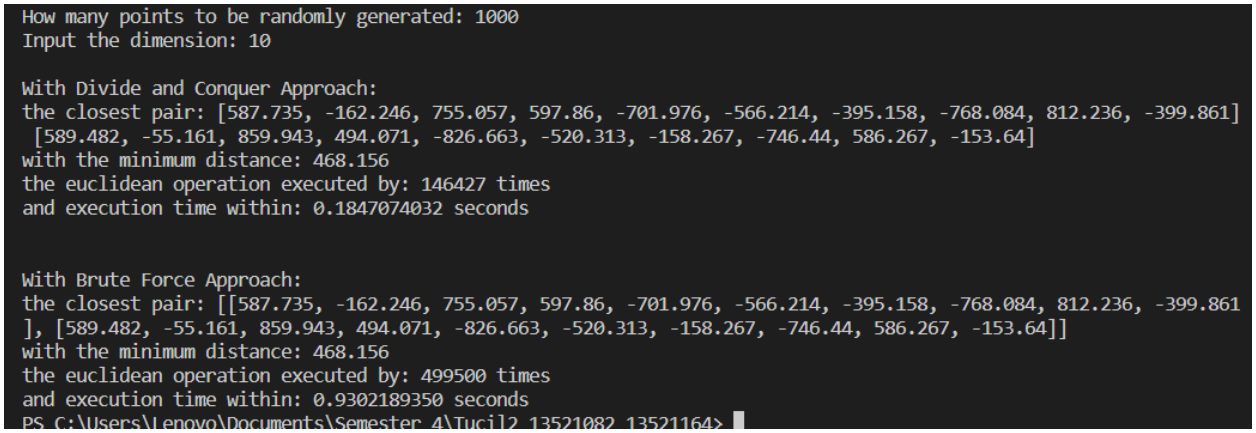
```
How many points to be randomly generated: 1000
Input the dimension: 3

With Divide and Conquer Approach:
the closest pair: [6068.361, -4214.528, -1850.618] [6185.956, -4190.945, -1865.441]
with the minimum distance: 120.849
the euclidean operation executed by: 2918 times
and execution time within: 0.0180439949 seconds


With Brute Force Approach:
the closest pair: [[6068.361, -4214.528, -1850.618], [6185.956, -4190.945, -1865.441]]
with the minimum distance: 120.849
the euclidean operation executed by: 499500 times
and execution time within: 0.4246420860 seconds
Do you want to visualize the points? (y/n):
```

Gambar 5.31 Hasil Pengujian 1000 Titik 3 Dimensi



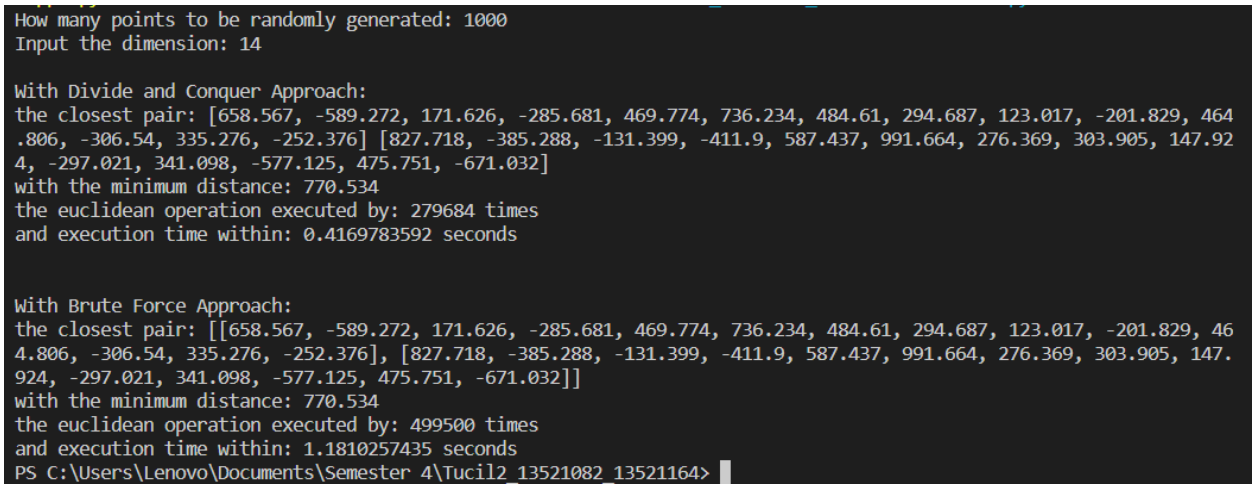
Gambar 5.32 Visualisasi 1000 Titik 3 Dimensi

d. Dimensi 10



Gambar 5.33 Hasil Pengujian 1000 Titik 10 Dimensi

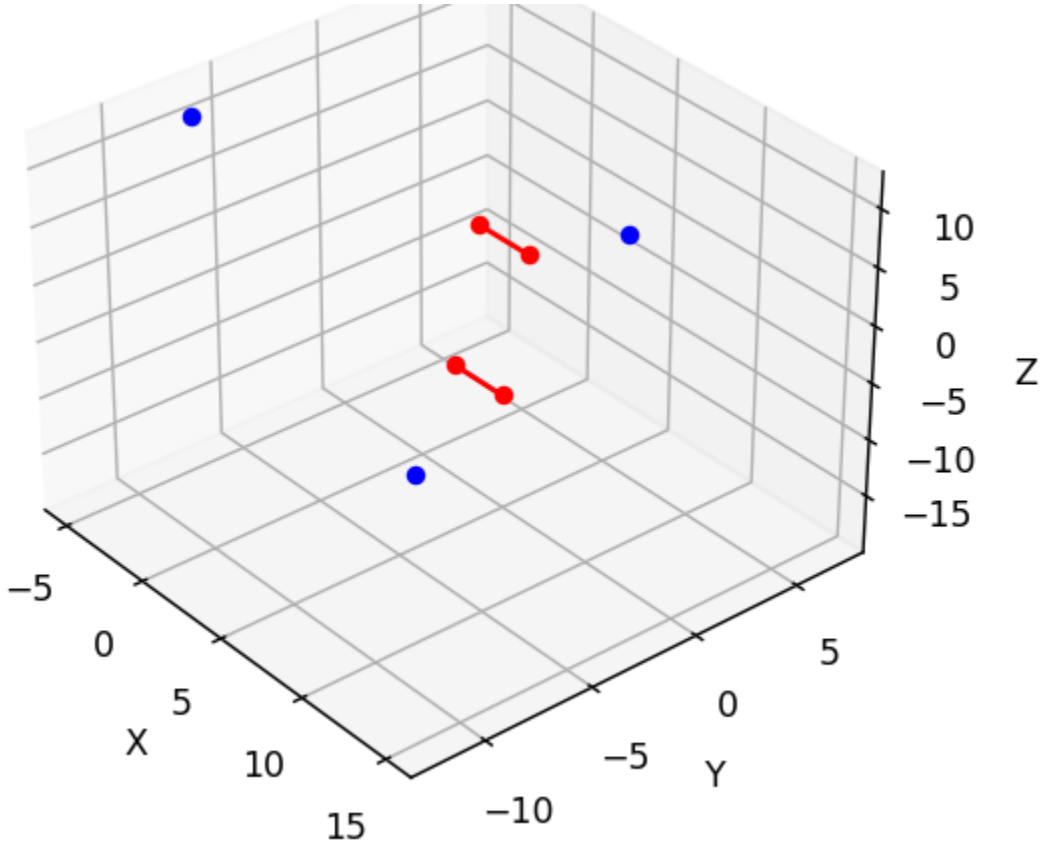
e. Dimensi 14



Gambar 5.34 Hasil Pengujian 1000 Titik 14 Dimensi

Kasus jika ada pasangan titik yang berjarak sama

(titik-titik untuk pengujian ini dibangkitkan secara manual karena sangat jarang sekali titik-titik yang dibangkitkan secara acak menampilkan pasangan titik yang berjarak sama lebih dari satu)



Gambar 5.35 Visualisasi Pasangan Titik Terdekat Lebih dari Satu Pasang

BAB VII
ANALISIS PENGUJIAN

Setelah dilakukan eksplorasi pengujian dengan berbagai titik dan berbagai dimensi, kami menemukan banyak hal menarik. Yang paling utama, algoritma *Divide and Conquer* benar memberikan hasil yang lebih efisien dibanding *Brute Force*, hal ini dibuktikan dari perbandingan banyaknya operasi euclidean antara keduanya serta waktu eksekusi yang menunjukkan algoritma *Divide and Conquer* cenderung lebih cepat menemukan solusi ketimbang *Brute Force*, terutama dengan input titik dan dimensi yang besar.

Kebenaran dari hasil algoritma *Divide and Conquer* dari program kami juga dapat dipastikan 100% benar karena ketika dibandingkan dengan hasil yang didapatkan dengan pendekatan *Brute Force* sudah sama persis, meskipun terkadang ada penulisan titik output berbeda urutannya, namun tidak mengubah fakta bahwa hasil kedua titik tersebut sama persis jika digambarkan pada bidang 3 dimensi.

LAMPIRAN

Link Repository Github:
farizkik/Tucil2_13521082_13521164: Tugas Kecil 2 untuk mata kuliah IF2211 Strategi Algoritma (github.com)

Ceklis Penilaian:

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