

**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

<b>DAFTAR ISI</b>	<b>1</b>
<b>BAB I DESKRIPSI TUGAS</b>	<b>2</b>
<b>BAB II LANDASAN TEORI</b>	<b>3</b>
<b>BAB III IMPLEMENTASI ALGORITMA DIVIDE AND CONQUER</b>	<b>4</b>
<b>BAB IV SOURCE CODE PROGRAM</b>	<b>7</b>
1. pointsGenerator.py	7
2. DNC.py	8
3. bruteForce.py	11
4. utilities.py	12
5. main.py	15
<b>BAB V HASIL PENGUJIAN</b>	<b>17</b>
1. Pengujian untuk $n = 16$	18
2. Pengujian untuk $n = 64$	23
3. Pengujian untuk $n = 128$	27
4. Pengujian untuk $n = 1000$	31
Kasus jika ada pasangan titik yang berjarak sama	35
<b>BAB VI ANALISIS PENGUJIAN</b>	<b>36</b>
<b>LAMPIRAN</b>	<b>37</b>

## 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 Tugasil 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 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  $\mathbb{R}^N$ , 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  $n \leq n_0$  then {ukuran persoalan  $P$  sudah cukup kecil}
  SOLVE persoalan  $P$  yang berukuran  $n$  ini
else
  DIVIDE menjadi  $r$  upa-persoalan,  $P_1, P_2, \dots, P_r$ , yang masing-masing berukuran  $n_1, n_2, \dots, n_r$ 
  for masing-masing  $P_1, P_2, \dots, P_r$ , do
    DIVIDEandCONQUER( $P_i, n_i$ )
  endfor
  COMBINE solusi dari  $P_1, P_2, \dots, P_r$  menjadi solusi persoalan semula
endif
```

Disadur dari: Homepage Rinaldi Munir, Strategi Algoritma

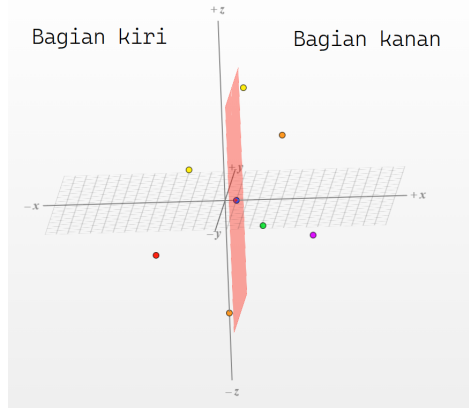
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/stima22-23.htm>

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  $P_1 = (x_1, y_1, z_1)$  dan  $P_2 = (x_2, y_2, z_2)$  dihitung dengan rumus Euclidean.

### BAB III

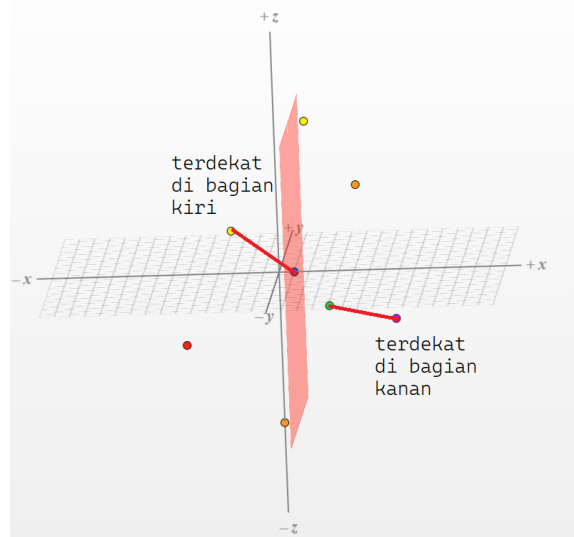
## IMPLEMENTASI ALGORITMA DIVIDE AND CONQUER

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

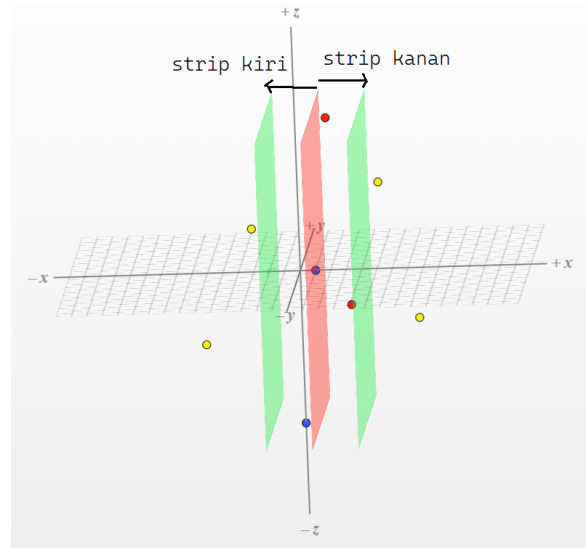
Masing-masing dari kedua bagian ini kemudian akan dicari pasangan titik terdekatnya dengan menggunakan fungsi pencarian pasangan titik terdekat yang sama. Ini adalah bagian *conquer* dari fungsi ini. Terdapat kasus basis berupa saat jumlah titik bernilai 1 dengan hasil keluaran tak hingga, dan saat jumlah 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

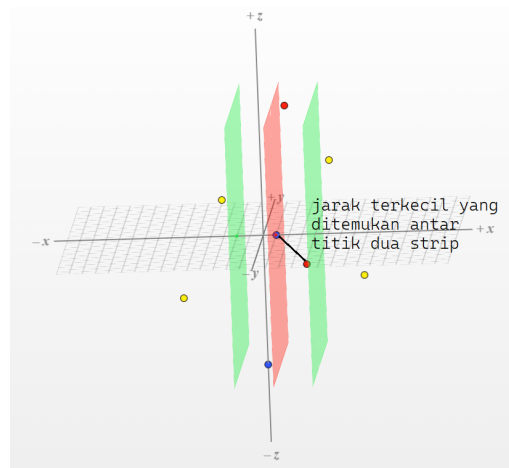
Setelah didapatkan jarak terkecil antara dua titik pada kedua bagian, maka akan dilakukan sebuah ‘penggabungan’ menggunakan algoritma *merge* antara kedua bagian tersebut. Di bagian ini, kita akan mencari pasangan titik dari bagian kanan dan bagian kiri yang memiliki jarak lebih kecil daripada jarak 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 `closestPairD()` akan memanggil fungsi `closestPairD()` 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  $O(n \log n)$ .

Untuk kasus dimensi secara general (dimensi  $\geq 2$ ), 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
1  import random
2
3  def generateRandomPoints():
4      # validasi input
5      while True:
6          n = input("How many points to be randomly generated: ") # number of points to generate
7          if (n.isdigit() and int(n) > 1):
8              n = int(n)
9              break
10         elif(n.isdigit() == False):
11             print("Input must be an integer\n")
12         else:
13             print("There's should be atleast 2 points to do so!\n")
14     while True:
15         N = input("Input the dimension: ") # number of dimension (in space R^N)
16         if (N.isdigit() and int(N) > 0):
17             N = int(N)
18             break
19         elif(N.isdigit() == False):
20             print("Input must be an integer\n")
21         else:
22             print("The dimension should be atleast 1\n")
23
24     points = []
25
26     for i in range(n):
27         point = []
28         for j in range(N):
29             # random float between 1000000 and 1000000
30             # array representing the point, rounded to 3 floating numbers
31             if(N<=5):
32                 point.append(round(random.uniform(-10000, 10000), 3))
33             else:
34                 point.append(round(random.uniform(-1000, 1000), 3))
35         points.append(point)
36     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.

```
1  from utilities import *
2
3  # Make strip given conditions
4  def stripMake(leftpoints,rightpoints,mid,sigma,gap=0):
5      leftstrip=[]
6      rightstrip=[]
7
8      for p in leftpoints:
9          if abs(p[gap]-mid)<=sigma:
10             leftstrip.append(p)
11      for p in rightpoints:
12          if abs(p[gap]-mid)<=sigma:
13             rightstrip.append(p)
14
15      leftstrip=mergeSort(leftstrip, 1,1)
16      rightstrip=mergeSort(rightstrip, 1,1)
17      return leftstrip,rightstrip
18
19 # divide points to left and right
20 def divide(points,gap=0):
21     N=len(points)
22
23     # sort points
24     points=mergeSort(points,1,gap)
25
26     # get midpoint
27     midpoint= points[N//2][gap]
28
29     # divide points to 2
30     leftpoints=points[:N//2]
31     rightpoints=points[N//2:]
32
33     return leftpoints,rightpoints,midpoint
34
```

```

34
35 # closest pair in 1d
36 def closestPair1D(points):
37     N=len(points)
38     points=mergeSort(points,1)
39     minDist=inf
40     closestPair=None
41     n_euclidean_computing=0
42     for i in range(N-1):
43         distance=euclideanDistance(points[i],points[i+1])
44         n_euclidean_computing+=1
45         if distance<minDist:
46             minDist=distance
47             closestPair=[points[i],points[i+1]]
48     return [minDist,closestPair,n_euclidean_computing]
49
50 # closest pair function
51 def closestPairnD(points):
52     N=len(points)
53     dimension=len(points[0])
54
55     # base case
56     if dimension==1:
57         return closestPair1D(points)
58     if N==1:
59         return [inf,None,0]
60     if N==2:
61         return [euclideanDistance(points[0],points[1]),[[points[0],points[1]]],1]
62
63     # if N>2, Divide and Conquer
64
65     # split points to 2
66     leftpoints,rightpoints,x0=divide(points)
67
68     # get closest dist in both sets of points, separately
69     n_euclidean_computing=0

```

```

70     d1=closestPairnD(leftpoints)
71     d2=closestPairnD(rightpoints)
72
73     n_euclidean_computing = d1[2]+d2[2]
74
75     # get closest dist out of the two
76     closestPair=None
77     d=inf
78     if(d1[0]<d2[0]):
79         d=d1[0]
80         closestPair=d1[1]
81     elif(d1[0]>d2[0]):
82         d=d2[0]
83         closestPair=d2[1]
84     else:
85         d=d1[0]
86         closestPair=d1[1]
87         for pairs in d2[1]:
88             closestPair.append(pairs)
89
90     #create strip
91     leftstrip,rightstrip=stripMake(leftpoints,rightpoints,x0,d)
92
93     # find ans
94     minDist=d
95     leftNstrip=len(leftstrip)
96     rightNstrip=len(rightstrip)
97

```

```

98     for i in range(leftNstrip):
99         for j in range(rightNstrip):
100             # sparsity condition,
101             # don't care if y distance exceed d
102             if rightstrip[j][1]-leftstrip[i][1]<-d:
103                 continue
104             if rightstrip[j][1]-leftstrip[i][1]>d:
105                 break
106             temp=euclideanDistance(leftstrip[i],rightstrip[j])
107             n_euclidean_computing+=1
108             if temp<minDist:
109                 minDist=temp
110                 closestPair=[[leftstrip[i],rightstrip[j]]]
111             elif temp==minDist:
112                 closestPair.append([leftstrip[i],rightstrip[j]])
113
114     # done, return minimum distance
115     return [minDist,closestPair,n_euclidean_computing]

```

**Gambar 4.2** Kode Program DNC.py

### 3. bruteForce.py

Sebagai pembandingan, 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
3  # Brute Force Algorithm for Finding Closest Pair of Points
4  def bruteForceClosestPair3D(points):
5      n_euclidean_computing = 0
6      n = len(points)
7      min_distance = inf
8      closest_pair = []
9      for i in range(n):
10         for j in range(i+1, n):
11             distance = sqrt((points[i][0]-points[j][0])**2 +
12                             (points[i][1]-points[j][1])**2 +
13                             (points[i][2]-points[j][2])**2)
14             n_euclidean_computing += 1
15             if (distance < min_distance):
16                 min_distance = distance
17                 closest_pair = []
18                 closest_pair.append([points[i], points[j]])
19             elif (distance == min_distance):
20                 closest_pair.append([points[i], points[j]])
21         return [min_distance, closest_pair, n_euclidean_computing]
22
23  def bruteForceClosestPairND(points):
24      n_euclidean_computing = 0
25      n = len(points)
26      min_distance = inf
27      closest_pair = None
28      for i in range(n):
29         for j in range(i+1, n):
30             distance = sqrt(sum([(points[i][k]-points[j][k])**2 for k in range(len(points[i]))]))
31             n_euclidean_computing += 1
32             if (distance < min_distance):
33                 min_distance = distance
34                 closest_pair = []
35                 closest_pair.append([points[i], points[j]])
36             elif (distance == min_distance):
37                 closest_pair.append([points[i], points[j]])
38         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
1  from math import *
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # Merge Sort Algorithm Using Divide and Conquer Approach for Array of Points
6  # in default, sorting would be ascending
7  def mergeSort(arr, asc = 1, el=0):
8      if len(arr) <= 1:
9          return arr
10
11     # Divide
12     mid = len(arr) // 2
13     left_arr = arr[:mid]
14     right_arr = arr[mid:]
15
16     left_arr = mergeSort(left_arr, asc, el)
17     right_arr = mergeSort(right_arr, asc, el)
18
19     # conquer
20     return merge(left_arr, right_arr, asc, el)
```

```
23 # conquer implementation
24 def merge(left_arr, right_arr, asc, el):
25     result = []
26     left_idx = 0
27     right_idx = 0
28
29     while left_idx < len(left_arr) and right_idx < len(right_arr):
30
31         # if ascending is True or 1
32         if (asc == 1):
33             if left_arr[left_idx][el] <= right_arr[right_idx][el]:
34                 result.append(left_arr[left_idx])
35                 left_idx += 1
36             else:
37                 result.append(right_arr[right_idx])
38                 right_idx += 1
39
40         # if ascending is False or any other than 1, hence descending
41         else:
42             if left_arr[left_idx][el] >= right_arr[right_idx][el]:
43                 result.append(left_arr[left_idx])
44                 left_idx += 1
45             else:
46                 result.append(right_arr[right_idx])
47                 right_idx += 1
48
49     if left_idx < len(left_arr):
50         result.extend(left_arr[left_idx:])
51     if right_idx < len(right_arr):
52         result.extend(right_arr[right_idx:])
53
54     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()
```

```

89 def visualize2D(points, closest_pair_list):
90     fig = plt.figure()
91     ax = plt.axes(projection='3d')
92     ax.set_xlabel('X')
93     ax.set_ylabel('Y')
94     ax.set_zlabel('Z')
95
96     for i in range(len(points)):
97         flag = False
98         for j in range(len(closest_pair_list)):
99             if points[i] in closest_pair_list[j]:
100                 flag = True
101         if not flag:
102             ax.scatter(points[i][0], points[i][1], c='b')
103         else:
104             ax.scatter(points[i][0], points[i][1], c='r')
105
106     for i in range(len(closest_pair_list)):
107         x = [closest_pair_list[i][0][0], closest_pair_list[i][1][0]]
108         y = [closest_pair_list[i][0][1], closest_pair_list[i][1][1]]
109         ax.plot(x, y, c='r')
110     plt.show()

```

```

111
112 v def visualize1D(points, closest_pair_list):
113     fig = plt.figure()
114     ax = plt.axes(projection='3d')
115     ax.set_xlabel('X')
116     ax.set_ylabel('Y')
117     ax.set_zlabel('Z')
118
119 v     for i in range(len(points)):
120         flag = False
121 v         for j in range(len(closest_pair_list)):
122 v             if points[i] in closest_pair_list[j]:
123                 flag = True
124 v         if not flag:
125             ax.scatter(points[i][0], 0, 0, c='b')
126 v         else:
127             ax.scatter(points[i][0], 0, 0, c='r')
128
129 v     for i in range(len(closest_pair_list)):
130         x = [closest_pair_list[i][0][0], closest_pair_list[i][1][0]]
131         ax.plot(x, [0, 0], [0, 0], c='r')
132     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 > ...
1  from DNC import *
2  from bruteForce import *
3  from pointsGenerator import *
4  from utilities import *
5  import time
6
7  # Generate Points
8  points, dimension = generateRandomPoints()
9
10 # Finding Closest Pair with Divide n Conquer Approach
11 start = time.time()
12 result = closestPairnD(points)
13 end = time.time()
14
15 print("\nWith Divide and Conquer Approach:")
16 print("the closest pair:", end=" ")
17 for i in result[1]:
18     print(i, end=' ')
19 print("\nwith the minimum distance:", round(result[0], 3))
20 print("the euclidean operation executed by:", result[2], "times")
21 print("and execution time within:", end=" ")
22 print("{0:.10f}".format(end-start), "seconds\n")
23
24 # Finding Closest Pair with Brute Force Approach
25 start = time.time()
26 result = bruteForceClosestPairnD(points)
27 end = time.time()
28
29 print("\nWith Brute Force Approach:")
30 print("the closest pair:", end=" ")
31 for i in result[1]:
32     print(i, end=' ')
33 print("\nwith the minimum distance:", round(result[0], 3))
34 print("the euclidean operation executed by:", result[2], "times")
35 print("and execution time within:", end=" ")
36 print("{0:.10f}".format(end-start), "seconds")
37
```



```

38 # Asking for visualization
39 if (dimension == 3):
40     while True:
41         visualize = input("Do you want to visualize the points? (y/n): ")
42         if (visualize == 'y'):
43             visualize3D(points, result[1])
44             break
45         elif (visualize == 'n'):
46             break
47         else:
48             print("Invalid input!")
49
50 elif (dimension == 2):
51     while True:
52         visualize = input("Do you want to visualize the points? (y/n): ")
53         if (visualize == 'y'):
54             visualize2D(points, result[1])
55             break
56         elif (visualize == 'n'):
57             break
58         else:
59             print("Invalid input!")
60
61 elif (dimension == 1):
62     while True:
63         visualize = input("Do you want to visualize the points? (y/n): ")
64         if (visualize == 'y'):
65             visualize1D(points, result[1])
66             break
67         elif (visualize == 'n'):
68             break
69         else:
70             print("Invalid input!")

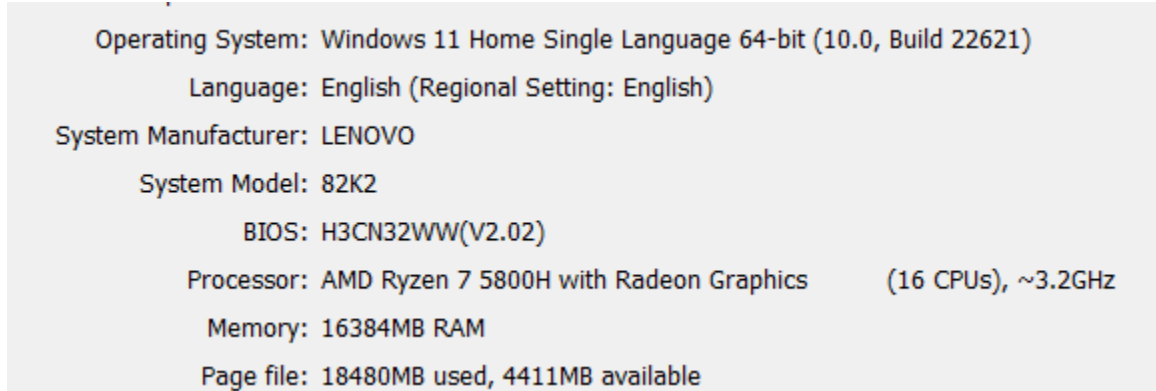
```

**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  $N$ .

```
How many points to be randomly generated: a
Input must be an integer

How many points to be randomly generated: 1
There's should be atleast 2 points to do so!

How many points to be randomly generated: 2
Input the dimension: a
Input must be an integer

Input the dimension: 0
The dimension should be atleast 1

Input the dimension: 3

With Divide and Conquer Approach:
the closest pair: [3059.84, -5151.818, 3903.928] [139
```

**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 $n = 16$

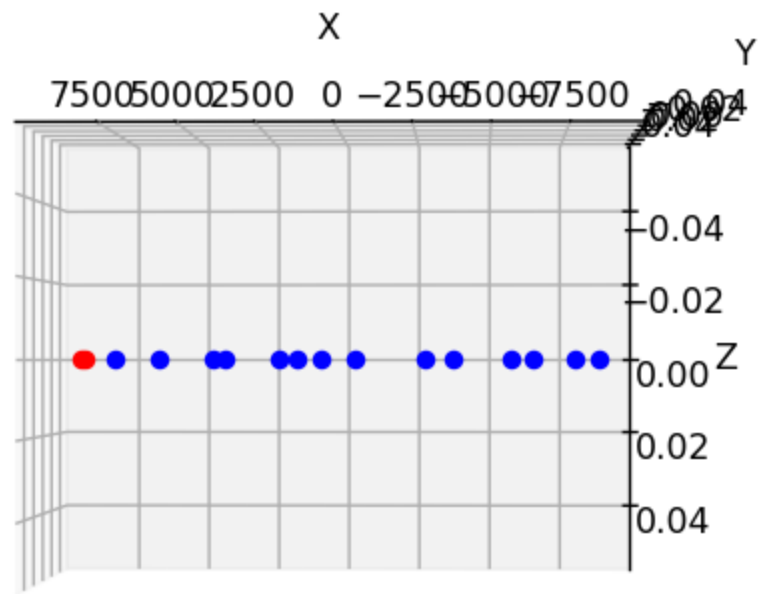
### a. Dimensi 1

```
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

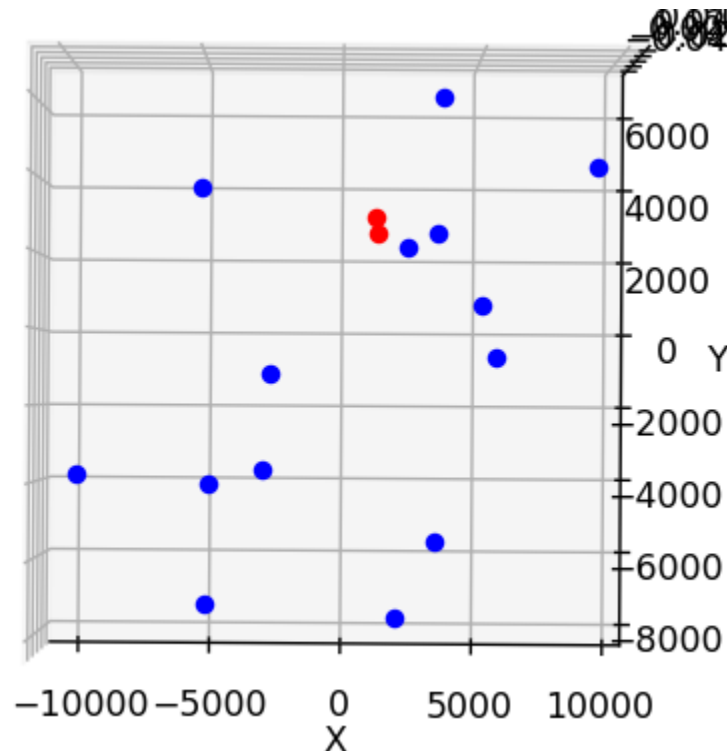
b. Dimensi 2

```
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

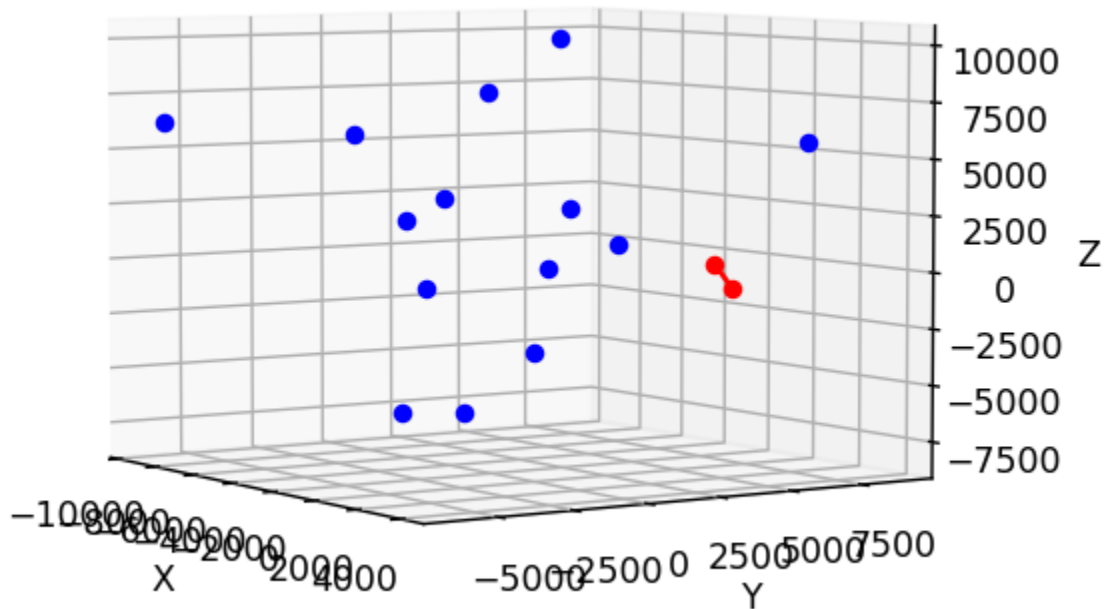
c. Dimensi 3

```
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: 0.000000000 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: 0.000000000 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 $n = 64$

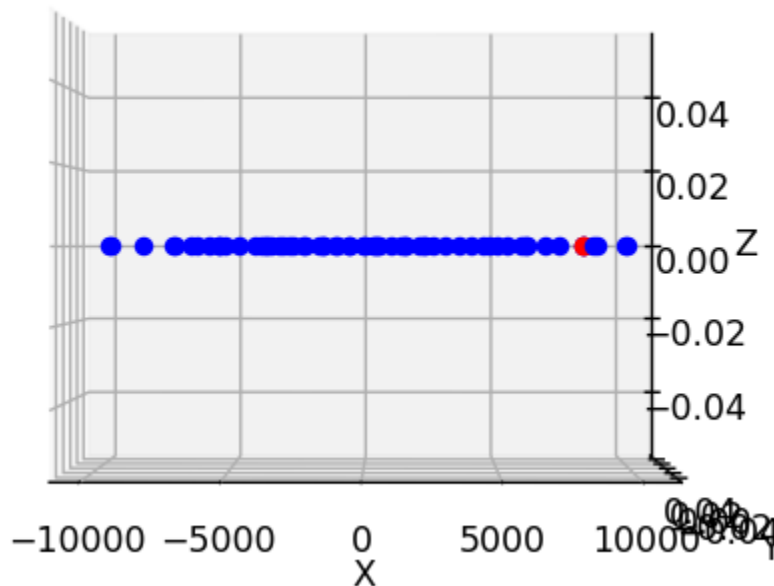
### a. Dimensi 1

```
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



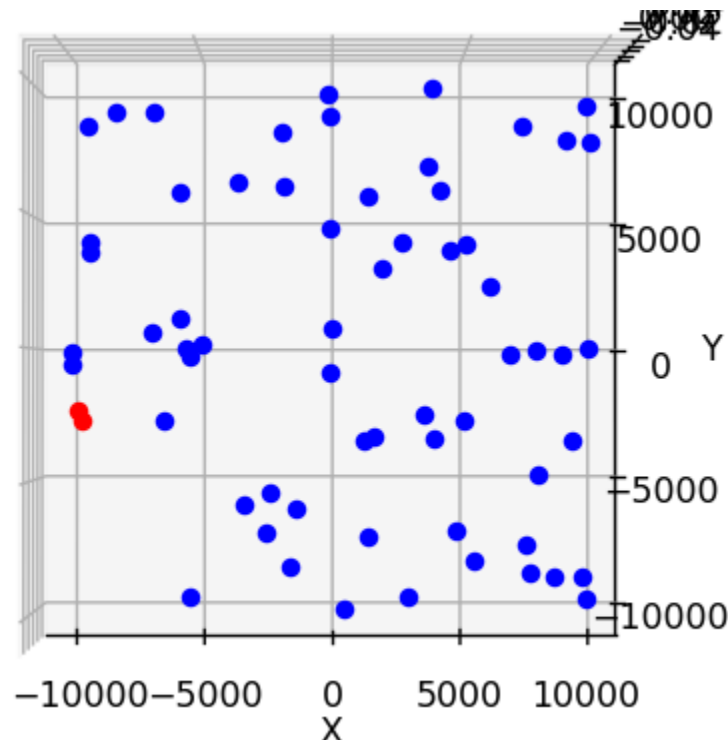
b. Dimensi 2

```
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

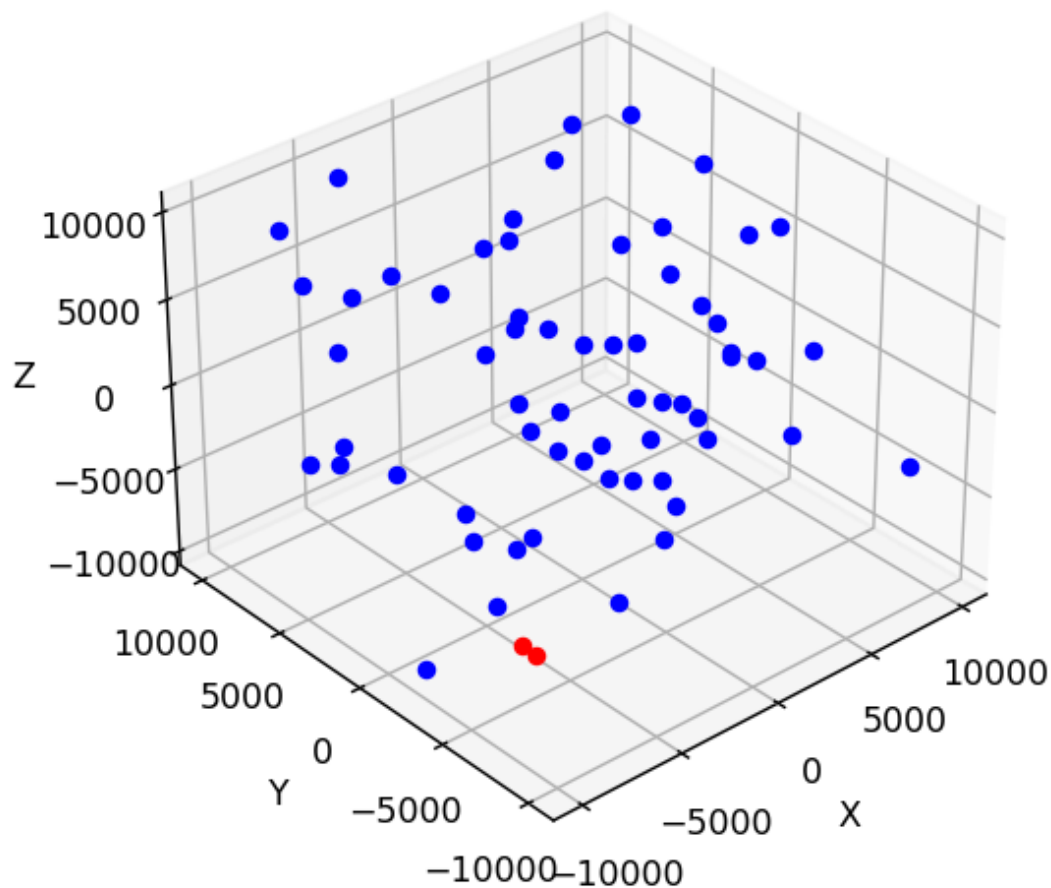
c. Dimensi 3

```
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
PS C:\Users\Lenovo\Documents\Semester 4\Tucil2_13521082_13521164>
```

**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, -107.911, -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 $n = 128$

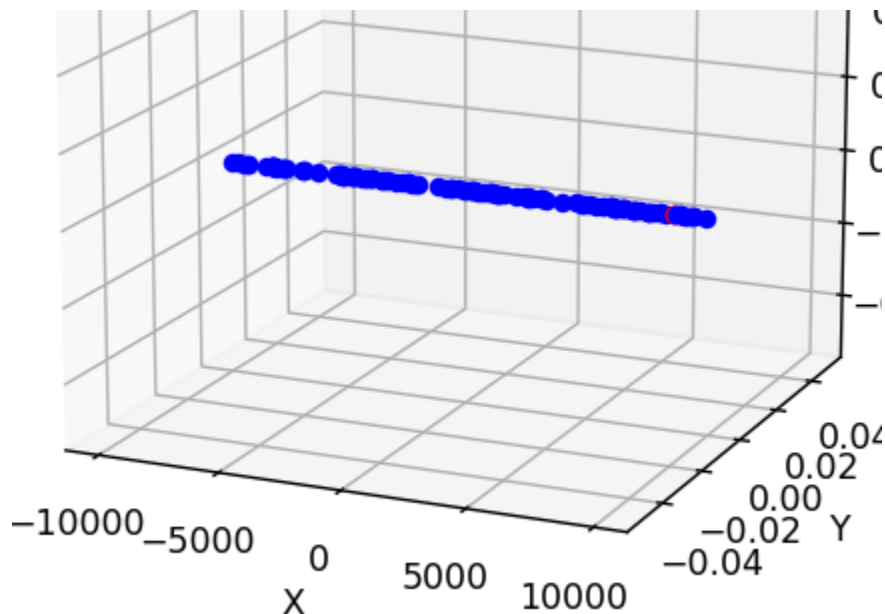
#### a. Dimensi 1

```
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

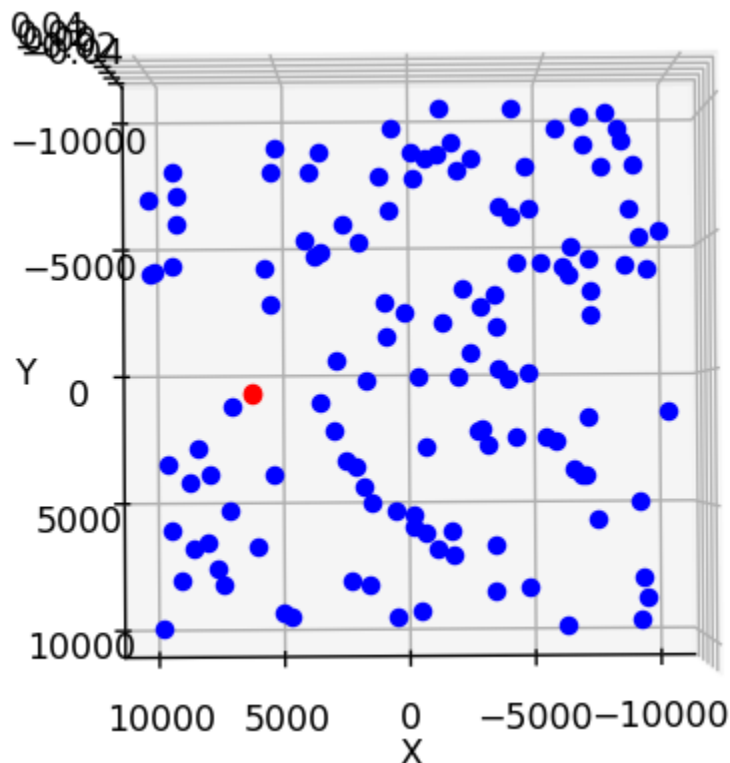
b. Dimensi 2

```
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

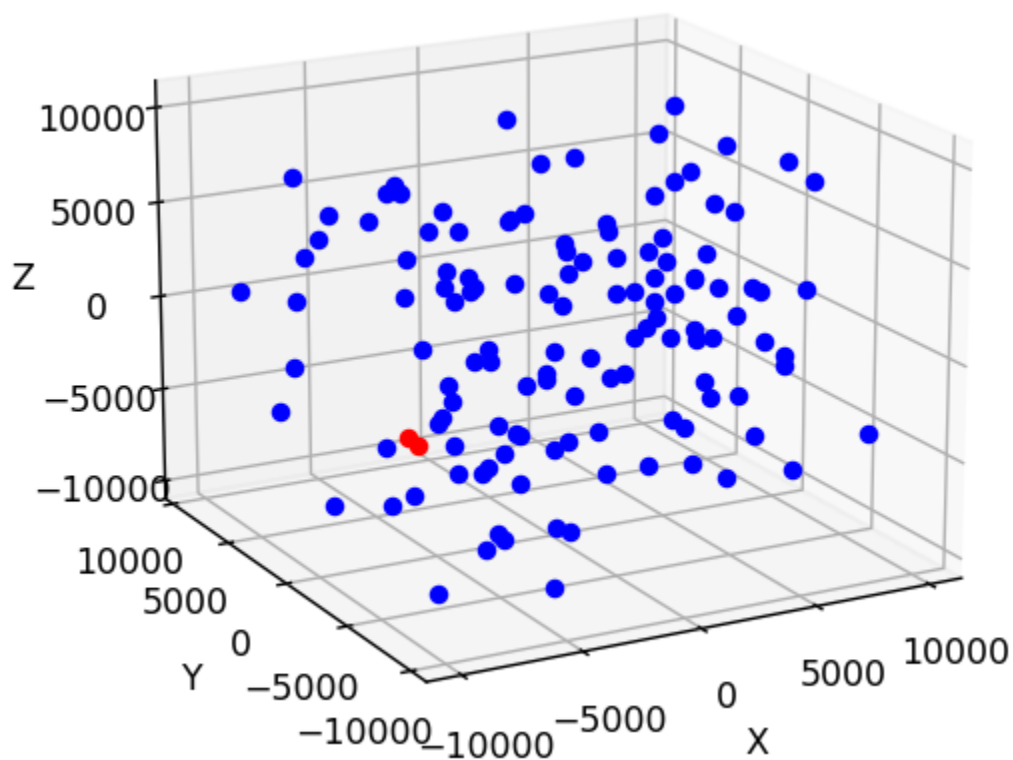
c. Dimensi 3

```
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 $n = 1000$

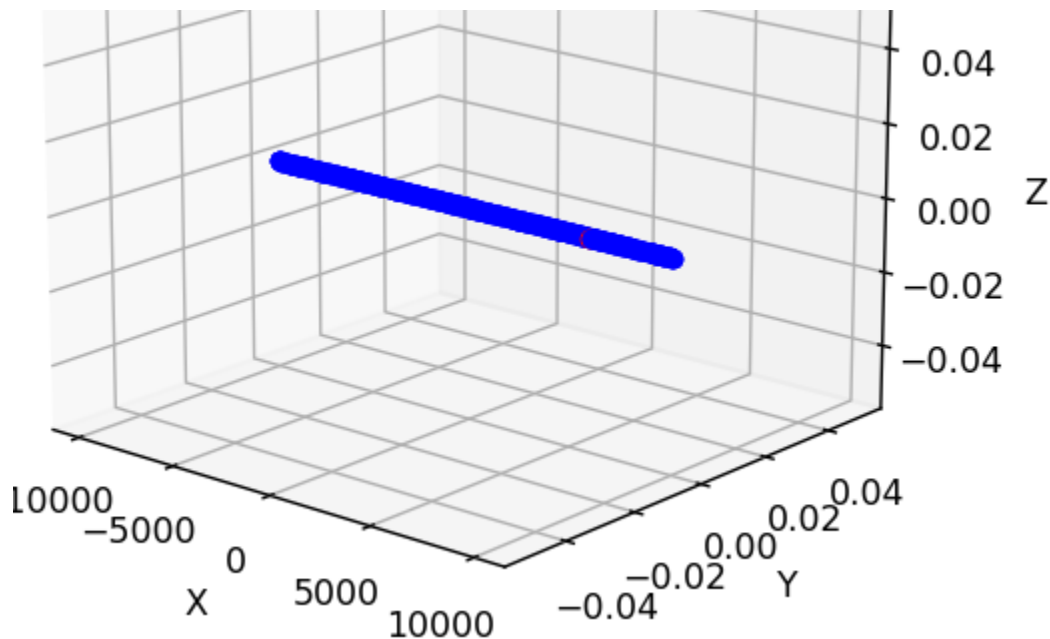
##### a. Dimensi 1

```
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



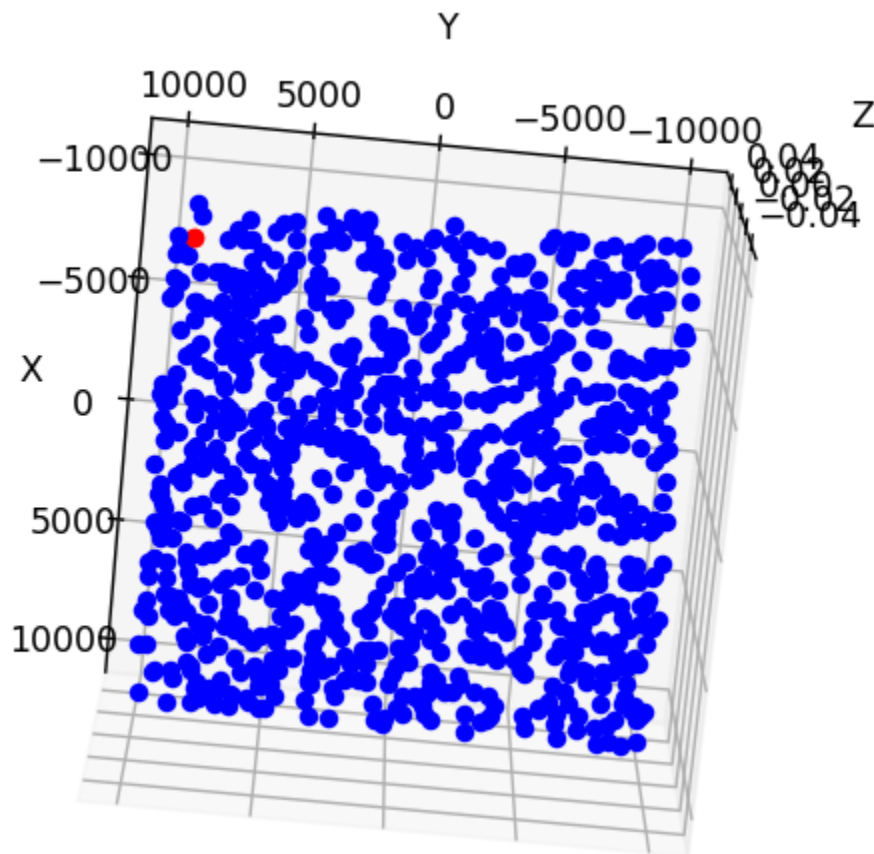
b. Dimensi 2

```
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

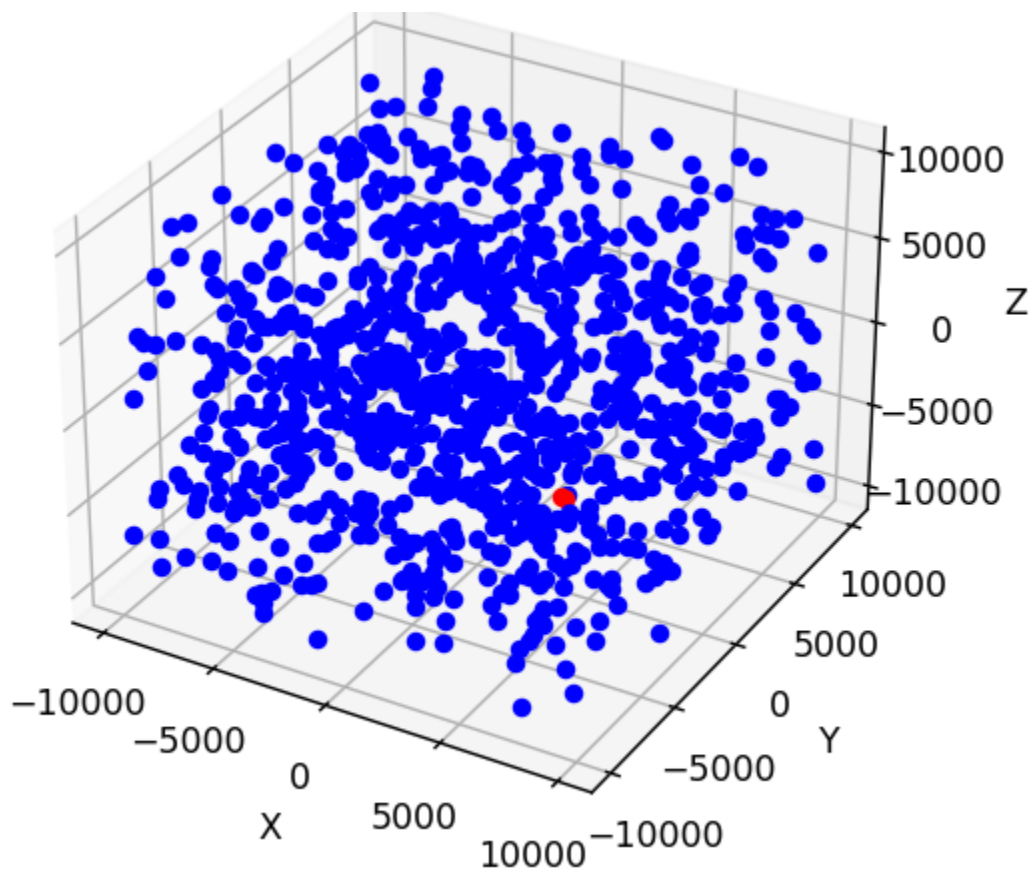
c. Dimensi 3

```
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

```
How many points to be randomly generated: 1000
Input the dimension: 10

With Divide and Conquer Approach:
the closest pair: [587.735, -162.246, 755.057, 597.86, -701.976, -566.214, -395.158, -768.084, 812.236, -399.861]
[589.482, -55.161, 859.943, 494.071, -826.663, -520.313, -158.267, -746.44, 586.267, -153.64]
with the minimum distance: 468.156
the euclidean operation executed by: 146427 times
and execution time within: 0.1847074032 seconds

With Brute Force Approach:
the closest pair: [[587.735, -162.246, 755.057, 597.86, -701.976, -566.214, -395.158, -768.084, 812.236, -399.861]
, [589.482, -55.161, 859.943, 494.071, -826.663, -520.313, -158.267, -746.44, 586.267, -153.64]]
with the minimum distance: 468.156
the euclidean operation executed by: 499500 times
and execution time within: 0.9302189350 seconds
PS C:\Users\Lenovo\Documents\Semester 4\Tucil2_13521082_13521164>
```

**Gambar 5.33** Hasil Pengujian 1000 Titik 10 Dimensi

e. Dimensi 14

```
How many points to be randomly generated: 1000
Input the dimension: 14

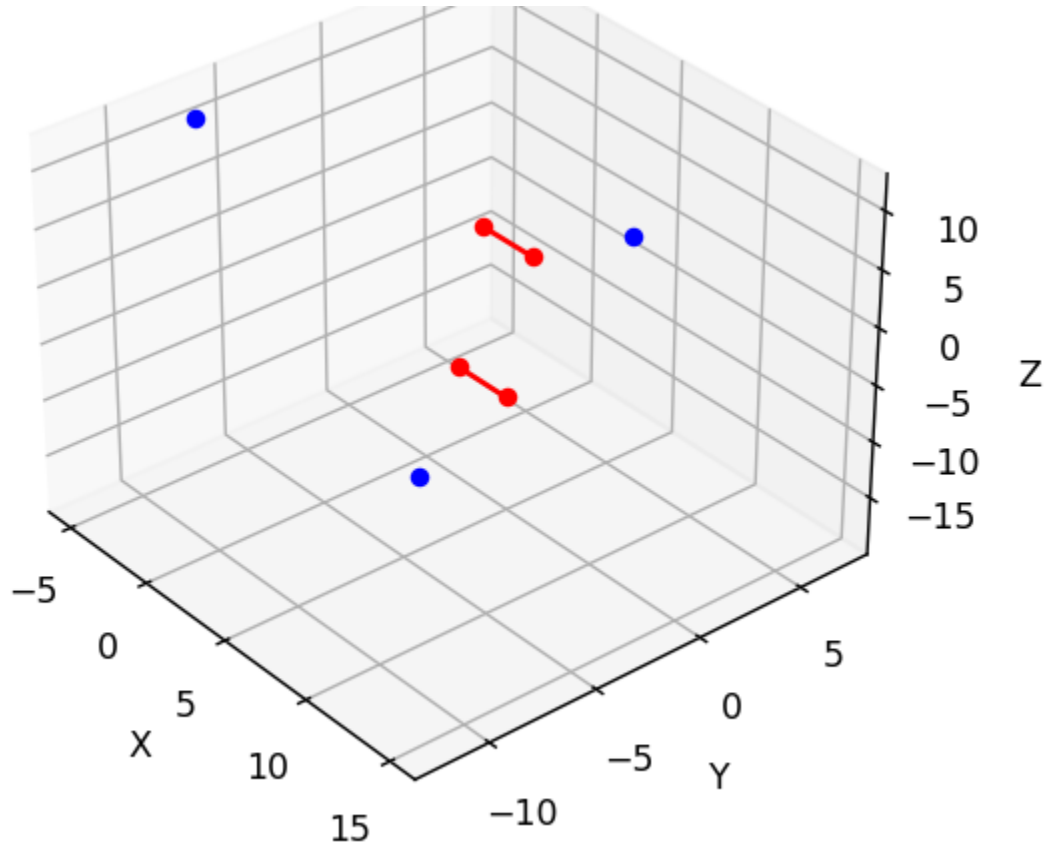
With Divide and Conquer Approach:
the closest pair: [658.567, -589.272, 171.626, -285.681, 469.774, 736.234, 484.61, 294.687, 123.017, -201.829, 464
.806, -306.54, 335.276, -252.376] [827.718, -385.288, -131.399, -411.9, 587.437, 991.664, 276.369, 303.905, 147.92
4, -297.021, 341.098, -577.125, 475.751, -671.032]
with the minimum distance: 770.534
the euclidean operation executed by: 279684 times
and execution time within: 0.4169783592 seconds

With Brute Force Approach:
the closest pair: [[658.567, -589.272, 171.626, -285.681, 469.774, 736.234, 484.61, 294.687, 123.017, -201.829, 46
4.806, -306.54, 335.276, -252.376], [827.718, -385.288, -131.399, -411.9, 587.437, 991.664, 276.369, 303.905, 147.
924, -297.021, 341.098, -577.125, 475.751, -671.032]]
with the minimum distance: 770.534
the euclidean operation executed by: 499500 times
and execution time within: 1.1810257435 seconds
PS C:\Users\Lenovo\Documents\Semester 4\Tucil2_13521082_13521164>
```

**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 VI 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 \(github.com\)](https://github.com/farizkik/Tucil2_13521082_13521164)

Ceklis Penilaian:

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