

LAPORAN TUGAS KECIL II

STRATEGI ALGORITMA

MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA DIVIDE AND CONQUER

Kelas Mahasiswa K03

Dosen: Ir. Rila Mandala, M.Eng., Ph.D.

**Diajukan sebagai tugas kecil Mata Kuliah IF2211 Strategi Algoritma pada Semester II
Tahun Akademik 2022/2023**



Disusun Oleh:

Jason Rivalino (13521008)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG**

2022

DAFTAR ISI

DAFTAR ISI.....	3
BAB I: DESKRIPSI MASALAH	4
BAB II: TEORI SINGKAT	5
I. Teori Singkat Algoritma <i>Divide and Conquer</i>	5
BAB III: PENJELASAN ALGORITMA DIVIDE AND CONQUER & SOURCE PROGRAM YANG DIBUAT	7
I. Penjelasan Algoritma Program yang Dibuat	7
a) Splash.py	8
b) Inputting.py	8
c) Projection.py	10
d) Processing.py	12
e) Main.py.....	16
BAB IV: EKSPERIMEN.....	19
I. Hasil Running Program	19
II. Uji Coba Hasil Program.....	22
BAB V: KESIMPULAN	26
REFERENSI.....	27
LAMPIRAN.....	28

BAB I

DESKRIPSI MASALAH

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

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

Program yang dibuat adalah 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

TEORI SINGKAT

I. Teori Singkat Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* merupakan algoritma yang berasal dari dua kata yaitu *Divide* dan *Conquer*. *Divide* pada persoalan algoritma ini sendiri adalah proses membagi persoalan menjadi *sub*-persoalan yang mirip dengan persoalan semula tetapi memiliki ukuran yang lebih kecil. Sedangkan, *conquer* dalam persoalan algoritma ini adalah proses untuk menyelesaikan masing-masing *sub*-persoalan. Penyelesaian yang dilakukan bisa secara langsung (untuk ukuran kecil) ataupun menggunakan proses rekursif (untuk ukuran besar). Setelah permasalahan diselesaikan, kemudian melakukan proses *combine* untuk menggabungkan solusi masing-masing *sub*-persoalan untuk membentuk solusi semula.

Dalam Algoritma *Divide and Conquer*, terdapat beberapa objek persoalan yang dibagi seperti tabel, matriks, eksponen, polinom, dll. Persoalan-persoalan ini pada umumnya memiliki karakteristik yang sama dengan karakteristik persoalan semula sehingga metode ini lebih cocok diselesaikan dengan skema rekursif.

Kompleksitas yang terdapat dalam Algoritma *Divide and Conquer* sendiri adalah sebagai berikut:

$$T(n) = \begin{cases} g(n), & n \leq n_o \\ T(n_1) + T(n_2) + \dots + T(n_r) + f(n), & n > n_o \end{cases}$$

Penjelasan:

$T(n)$ = kompleksitas waktu penyelesaian persoalan P berukuran n

$g(n)$ = kompleksitas waktu untuk proses *solving* jika n sudah berukuran kecil

$T(n_1) + T(n_2) + \dots + T(n_r)$ = kompleksitas waktu untuk memproses setiap *sub*-persoalan

$f(n)$ = kompleksitas waktu untuk proses *combine* solusi dari setiap masing-masing *sub*-persoalan

Beberapa contoh dari penerapan Algoritma *Brute Force* ini adalah sebagai berikut:

1. Persoalan pencarian nilai maksimum dan minimum (MinMaks)
2. Persoalan menghitung perpangkatan

IF2211**Strategi Algoritma**

3. Persoalan mengevaluasi pohon ekspresi
4. Persoalan pengurutan (*sorting*)
5. Teorema Master
6. Mencari pasangan titik terdekat (*Closest Pair*)
7. Perkalian matriks
8. Perkalian bilangan bulat yang besar
9. Perkalian polinom
10. *Convex Hull*

BAB III

PENJELASAN ALGORITMA *DIVIDE AND CONQUER* & SOURCE PROGRAM YANG DIBUAT

I. Penjelasan Algoritma Penyelesaian Masalah

Untuk tahapan dari penyelesaian masalah *Closest Pair* dengan algoritma *Divide and Conquer* adalah sebagai berikut:

1. Mengurutkan tiap titik koordinat menaik berdasarkan nilai absisnya (pra-proses)
2. Menghitung basis untuk kondisi titik koordinat bernilai dua atau tiga secara langsung dengan rumus *Euclidean*
3. Jika titik ada lebih dari tiga, melakukan proses pembagian titik menjadi himpunan dua bagian dengan jumlah titik yang sama, pembagian kedua titik ini dibatasi oleh sebuah garis imaginasi
4. Melakukan proses penyelesaian permasalahan secara rekursif dengan algoritma *Divide and Conquer* untuk mencari sepasang titik terdekat
5. Jika kondisi titik terdekat berada dalam bagian kiri atau kanan garis imaginasi, solusi ditemukan
6. Ada kemungkinan titik terdekat berada pada posisi yang dipisahkan oleh garis imajiner. Jika demikian, membuat daerah strip sebesar dua kali nilai minimal dari solusi langkah 5 dan menentukan koordinat-koordinat yang termasuk didalam daerah strip
7. Untuk koordinat yang ada dalam daerah strip, urutkan berdasarkan nilai ordinatnya
8. Menghitung jarak yang ada dalam strip dan membandingkannya dengan solusi yang didapat dari langkah 5

II. Penjelasan Algoritma Program yang Dibuat

Pada program yang saya buat, program terdiri dari beberapa file yang tersusun atas satu file utama (*main file*) yaitu *main.py* dan terdapat beberapa file yang terdiri dari berbagai macam fungsi untuk menjalankan program utamanya antara lain *splash.py*, *inputting.py*, *projection.py*, dan *processing.py*.

a) Splash.py

Program pada file ini berfungsi untuk menampilkan splash screen saat *opening* dan *closing* dari program yang dijalankan. Deskripsi dari fungsi yang terdapat dalam file ini antara lain:

Nama Fungsi/Prosedur	Deskripsi
def openingSplash()	Splash Screen untuk ditampilkan saat membuka aplikasi
def closingSplash()	Splash Screen untuk ditampilkan saat menutup aplikasi

```

1  def openingSplash():
2      print("
3      print("
4      print("
5      print("
6      print("
7      print("
8      print("
9      print("
10     print("
11     print("
12     print("
13     print("
14     print("
15     print("
16     print("
17     print("
18     print("
19     print("-----")
20     print("Selamat datang di program pencarian Closest Pair")
21     print("-----")
22
23 def closingSplash():
24     print ("
25     print ("*****")
26     print ("
27     print ("
28     print ("
29     print ("
30     print ("
31     print ("
32     print ("
33     print ("*****")
34     print ("
35     print ("-----")
36     print ("Terima kasih telah menggunakan program ini !!")
37     print ("-----")
38     print ("

```

b) Inputting.py

Program pada file ini berfungsi untuk melakukan proses input jumlah titik, ukuran dimensi, dan juga letak koordinat titik-titik yang akan dicari jarak terdekatnya. Pada program input ini, untuk input titik, hanya dibatasi antara 2 hingga 1000 titik. Untuk input dimensi juga hanya terdapat opsi untuk dua dan tiga dimensi saja. Pencarian titik koordinat dilakukan dengan menggunakan pembangkit bilangan acak.

Deskripsi dari fungsi yang terdapat dalam file ini antara lain:

Nama Fungsi/Prosedur	Deskripsi
def inputTitik(n)	Fungsi untuk melakukan input jumlah titik yang akan dibandingkan
def inputDimension(n)	Fungsi untuk melakukan input dimensi yang ingin dipilih
def inputKoordinat(n, dimensi)	Fungsi untuk melakukan input koordinat secara acak (bisa untuk satu hingga tiga dimensi)

```

1  import numpy as np
2  from numpy import random
3
4  # Melakukan input jumlah titik yang ingin dihitung
5  def inputTitik (n):
6      n = int(input("Masukkan jumlah titik (2-1000): "))
7      while (n < 2 or n > 1000):
8          n = int(input("Masukkan jumlah titik (2-1000): "))
9      return n
10
11 # Melakukan input dimensi
12 def inputDimension (n):
13     print("Opsi Pilihan:")
14     print("1) Mencari titik pada 1 dimensi")
15     print("2) Mencari titik pada 2 dimensi")
16     print("3) Mencari titik pada 3 dimensi")
17     n = int(input("Masukkan pilihan dimensi: "))
18     while (n != 1 and n != 2 and n != 3):
19         n = int(input("Masukkan pilihan dimensi: "))
20     return n
21
22 # Melakukan input koordinat dengan pembangkit bilangan acak
23 def inputKoordinat(n,dimensi):
24     coordinates = []
25     for i in range (n):
26         if (dimensi == 1):
27             x = random.randint(1,1000)
28             coordinates.append([x])
29         elif (dimensi == 2):
30             x = random.randint(1,250)
31             y = random.randint(1,250)
32             coordinates.append([x,y])
33         elif (dimensi == 3):
34             x = random.randint(1,250)
35             y = random.randint(1,250)
36             z = random.randint(1,250)
37             coordinates.append([x,y,z])
38     return coordinates

```

c) Projection.py

Program pada file ini berfungsi untuk memproses koordinat yang sudah diinput sebelumnya untuk diproyeksikan kedalam bentuk gambar visual.

Deskripsi dari fungsi yang terdapat dalam file ini antara lain:

Nama Fungsi/Prosedur	Deskripsi
def projectionBefore (listKoordinat,dimensi)	Fungsi untuk menentukan fungsi proyeksi mana yang akan dijalankan berdasarkan dimensinya (menentukan fungsi yang sebelum mencari titik terdekat)
def projectionAfter(listKoordinat, listKoordinatTerdekat, dimensi)	Fungsi untuk menentukan fungsi proyeksi mana yang akan dijalankan berdasarkan dimensinya (menentukan fungsi setelah ditemukan titik terdekat)
def projectionBefore1D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat sebelum pencarian jarak titik terdekat (tampilan dalam 1D)
def projectionBefore2D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat sebelum pencarian jarak titik terdekat (tampilan dalam 2D)
def projectionBefore3D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat sebelum pencarian jarak titik terdekat (tampilan dalam 3D)
def projectionAfter1D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat setelah pencarian jarak titik terdekat beserta dengan dua titik terdekatnya (tampilan dalam 1D)
def projectionAfter2D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat setelah pencarian jarak titik terdekat beserta dengan dua titik terdekatnya (tampilan dalam 2D)
def projectionAfter3D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat setelah pencarian jarak titik terdekat

	beserta dengan dua titik terdekatnya (tampilan dalam 3D)
--	--

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from mpl_toolkits import mplot3d
4
5  # Fungsi untuk menampilkan proyeksi awal
6  def projectionBefore(listKoordinat, dimensi):
7      if dimensi == 1:
8          projectionBefore1D(listKoordinat)
9      elif dimensi == 2:
10         projectionBefore2D(listKoordinat)
11     elif dimensi == 3:
12         projectionBefore3D(listKoordinat)
13
14     # Fungsi untuk menampilkan proyeksi jarak titik terdekat
15     def projectionAfter(listKoordinat, listKoordinatTerdekat, dimensi):
16         if dimensi == 1:
17             projectionAfter1D(listKoordinat, listKoordinatTerdekat)
18         elif dimensi == 2:
19             projectionAfter2D(listKoordinat, listKoordinatTerdekat)
20         elif dimensi == 3:
21             projectionAfter3D(listKoordinat, listKoordinatTerdekat)
22
```

```
23 # Fungsi proyeksi awal 1D
24 def projectionBefore1D(listKoordinat):
25     # Data untuk proyeksi 1D
26     for i in range (len(listKoordinat)):
27         plt.scatter(listKoordinat[i][0], 0, color='red')
28     print("Menampilkan Proyeksi Titik Koordinat Awal...")
29     plt.show()
30
```

```
31 # Fungsi proyeksi awal 2D
32 def projectionBefore2D(listKoordinat):
33     # Data untuk proyeksi 2D
34     for i in range (len(listKoordinat)):
35         plt.scatter(listKoordinat[i][0], listKoordinat[i][1], color='red')
36     print("Menampilkan Proyeksi Titik Koordinat Awal...")
37     plt.show()
38
39 # Fungsi proyeksi awal 3D
40 def projectionBefore3D(listKoordinat):
41     axes = plt.axes(projection='3d')
42
43     # Data untuk proyeksi 3D
44     for i in range (len(listKoordinat)):
45         np.linspace(listKoordinat[i][0], listKoordinat[i][1], listKoordinat[i][2])
46
47         # Plotting titik-titik
48         axes.scatter3D(listKoordinat[i][0], listKoordinat[i][1], listKoordinat[i][2], color='red')
49
50     print("Menampilkan Proyeksi Titik Koordinat Awal...")
51     plt.show()
52
```

```

53 # Fungsi proyeksi jarak titik terdekat 1D
54 def projectionAfter1D(listKoordinat, listKoordinatTerdekat):
55     for i in range (len(listKoordinat)):
56         plt.scatter(listKoordinat[i][0], 0, color='red')
57
58     for i in range (len(listKoordinatTerdekat)):
59         plt.scatter(listKoordinatTerdekat[i][0], 0, color='blue')
60
61     print("Menampilkan Proyeksi Titik Koordinat Terdekat...")
62     print("=====")
63     plt.show()
64
65 # Fungsi proyeksi jarak titik terdekat 2D
66 def projectionAfter2D(listKoordinat, listKoordinatTerdekat):
67     for i in range (len(listKoordinat)):
68         plt.scatter(listKoordinat[i][0], listKoordinat[i][1], color='red')
69
70     for i in range (len(listKoordinatTerdekat)):
71         plt.scatter(listKoordinatTerdekat[i][0], listKoordinatTerdekat[i][1], color='blue')
72
73     print("Menampilkan Proyeksi Titik Koordinat Terdekat...")
74     print("=====")
75     plt.show()
76
77 # Fungsi proyeksi jarak titik terdekat 3D
78 def projectionAfter3D(listKoordinat, listKoordinatTerdekat):
79     axes = plt.axes(projection='3d')
80
81     # Data untuk proyeksi 3D
82     for i in range (len(listKoordinat)):
83         np.linspace(listKoordinat[i][0], listKoordinat[i][1], listKoordinat[i][2])
84
85     # Plotting titik-titik
86     axes.scatter3D(listKoordinat[i][0], listKoordinat[i][1], listKoordinat[i][2], color='red')
87
88     # Plotting garis antara titik terdekat
89     for i in range (len(listKoordinatTerdekat)):
90         axes.scatter3D(listKoordinatTerdekat[i][0], listKoordinatTerdekat[i][1], listKoordinatTerdekat[i][2], color='blue')
91
92     print("Menampilkan Proyeksi Titik Koordinat Terdekat...")
93     print("=====")
94     plt.show()

```

d) Processing.py

Program pada file ini berfungsi untuk melakukan proses pencarian titik terdekat. Beberapa fungsi yang terdapat dalam program ini antara lain fungsi untuk melakukan *sorting* terurut menaik berdasarkan koordinat titik. Lalu ada fungsi untuk melakukan *brute force* dan juga untuk *divide and conquer*. Selain itu, ada juga fungsi untuk menampilkan spesifikasi *device* yang dipergunakan untuk melakukan *running* program.

Deskripsi dari fungsi yang terdapat dalam file ini antara lain:

Nama Fungsi/Prosedur	Deskripsi
def sortingX(listKoordinat)	Fungsi untuk melakukan <i>sorting</i> koordinat terurut berdasarkan nilai x (absis)
def sortingY(listKoordinat)	Fungsi untuk melakukan <i>sorting</i> koordinat terurut berdasarkan nilai y (ordinat)
def bruteForce (listKoordinat,dimensi)	Fungsi untuk melakukan pencarian titik terdekat dengan algoritma <i>brute force</i>

IF2211
Strategi Algoritma

def divideAndConquer (listKoordinat,n,dimensi)	Fungsi untuk melakukan pencarian titik terdekat dengan algoritma <i>divide and conquer</i>
def spekLaptop()	Fungsi untuk menampilkan spesifikasi laptop yang akan digunakan untuk <i>running</i> program

```

1  import math
2  import psutil
3  import platform
4
5  # Sorting untuk mengurutkan koordinat berdasarkan x menaik
6  def sortingX(listKoordinat):
7      for i in range (len(listKoordinat)):
8          for j in range (len(listKoordinat)):
9              if listKoordinat[i][0] < listKoordinat[j][0]:
10                 listKoordinat[i], listKoordinat[j] = listKoordinat[j], listKoordinat[i]
11      return listKoordinat
12
13 # Sorting untuk mengurutkan koordinat berdasarkan y menaik
14 def sortingY(listKoordinat):
15     for i in range (len(listKoordinat)):
16         for j in range (len(listKoordinat)):
17             if listKoordinat[i][1] < listKoordinat[j][1]:
18                 listKoordinat[i], listKoordinat[j] = listKoordinat[j], listKoordinat[i]
19     return listKoordinat
20

```

```

21 #inisialisasi variabel global
22 hitungEuclideanBF = 0
23
24 # Melakukan proses brute force untuk mencari jarak terdekat
25 def bruteForce(listKoordinat,dimensi):
26     global hitungEuclideanBF
27     jarak = []
28     min = float("inf")
29     for i in range (len(listKoordinat)):
30         for j in range (len(listKoordinat)):
31             if i != j:

```

```

32                 if dimensi == 1:
33                     jarak.append(math.sqrt((pow((listKoordinat[j][0]-listKoordinat[i][0]),2))))
34                     hitungEuclideanBF += 1
35                 elif dimensi == 2:
36                     jarak.append(math.sqrt((pow((listKoordinat[j][0]-listKoordinat[i][0]),2))+
37                                             (pow((listKoordinat[j][1]-listKoordinat[i][1]),2))))
38                     hitungEuclideanBF += 1
39                 elif dimensi == 3:
40                     jarak.append(math.sqrt((pow((listKoordinat[j][0]-listKoordinat[i][0]),2))+
41                                             (pow((listKoordinat[j][1]-listKoordinat[i][1]),2))+
42                                             (pow((listKoordinat[j][2]-listKoordinat[i][2]),2))))
43                     hitungEuclideanBF += 1
44     for i in range (len(jarak)):
45         if jarak[i] < min:
46             min = jarak[i]
47     return min
48

```

```
49 #inisialisasi variabel global
50 hitungEuclideanDC = 0
51
52 # Melakukan proses divide and conquer untuk mencari jarak terdekat
53 def divideAndConquer(listKoordinat, n, dimensi):
54     global hitungEuclideanDC
55     n = len(listKoordinat)
56     jarak = 0
57     # Basis 1
58     if n == 2:
59         if dimensi == 1:
60             jarak = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2)))
61             hitungEuclideanDC += 1
62             tempNearest = [listKoordinat[0], listKoordinat[1]]
63             return jarak, tempNearest
64         elif dimensi == 2:
65             jarak = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2))+
66                               (pow((listKoordinat[1][1]-listKoordinat[0][1]),2)))
67             hitungEuclideanDC += 1
68             tempNearest = [listKoordinat[0], listKoordinat[1]]
69             return jarak, tempNearest
70         elif dimensi == 3:
71             jarak = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2))+
72                               (pow((listKoordinat[1][1]-listKoordinat[0][1]),2))+
73                               (pow((listKoordinat[1][2]-listKoordinat[0][2]),2)))
74             hitungEuclideanDC += 1
75             tempNearest = [listKoordinat[0], listKoordinat[1]]
76             return jarak, tempNearest
```

```
77 # Basis 2
78 elif n == 3:
79     if dimensi == 1:
80         jarak1 = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2)))
81         jarak2 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[0][0]),2)))
82         jarak3 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[1][0]),2)))
83         if jarak1 < jarak2 and jarak1 < jarak3:
84             hitungEuclideanDC += 1
85             jarak = jarak1
86             tempNearest = [listKoordinat[0], listKoordinat[1]]
87         elif jarak2 < jarak1 and jarak2 < jarak3:
88             hitungEuclideanDC += 1
89             jarak = jarak2
90             tempNearest = [listKoordinat[0], listKoordinat[2]]
91         else:
92             hitungEuclideanDC += 1
93             jarak = jarak3
94             tempNearest = [listKoordinat[1], listKoordinat[2]]
95     return jarak, tempNearest
```

```

96     elif dimensi == 2:
97         jarak1 = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2))+
98                             (pow((listKoordinat[1][1]-listKoordinat[0][1]),2)))
99         jarak2 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[0][0]),2))+
100                             (pow((listKoordinat[2][1]-listKoordinat[0][1]),2)))
101         jarak3 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[1][0]),2))+
102                             (pow((listKoordinat[2][1]-listKoordinat[1][1]),2)))
103         if jarak1 < jarak2 and jarak1 < jarak3:
104             hitungEuclideanDC += 1
105             jarak = jarak1
106             tempNearest = [listKoordinat[0], listKoordinat[1]]
107         elif jarak2 < jarak1 and jarak2 < jarak3:
108             hitungEuclideanDC += 1
109             jarak = jarak2
110             tempNearest = [listKoordinat[0], listKoordinat[2]]
111         else:
112             hitungEuclideanDC += 1
113             jarak = jarak3
114             tempNearest = [listKoordinat[1], listKoordinat[2]]
115         return jarak, tempNearest

116     elif dimensi == 3:
117         jarak1 = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2))+
118                             (pow((listKoordinat[1][1]-listKoordinat[0][1]),2))+
119                             (pow((listKoordinat[1][2]-listKoordinat[0][2]),2)))
120         jarak2 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[0][0]),2))+
121                             (pow((listKoordinat[2][1]-listKoordinat[0][1]),2))+
122                             (pow((listKoordinat[2][2]-listKoordinat[0][2]),2)))
123         jarak3 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[1][0]),2))+
124                             (pow((listKoordinat[2][1]-listKoordinat[1][1]),2))+
125                             (pow((listKoordinat[2][2]-listKoordinat[1][2]),2)))
126         if jarak1 < jarak2 and jarak1 < jarak3:
127             jarak = jarak1
128             tempNearest = [listKoordinat[0], listKoordinat[1]]
129             # print(tempNearest)
130             hitungEuclideanDC += 1
131         elif jarak2 < jarak1 and jarak2 < jarak3:
132             jarak = jarak2
133             tempNearest = [listKoordinat[0], listKoordinat[2]]
134             # print(tempNearest)
135             hitungEuclideanDC += 1
136         else:
137             jarak = jarak3
138             tempNearest = [listKoordinat[1], listKoordinat[2]]
139             # print(tempNearest)
140             hitungEuclideanDC += 1
141         return jarak, tempNearest

142     # Rekurens
143     else:
144         # Membagi data menjadi 2 bagian
145         middle = n//2
146         left_area = []
147         right_area = []
148         for i in range (middle):
149             left_area.append(listKoordinat[i])
150         for i in range (middle, n):
151             right_area.append(listKoordinat[i])
152
153         # Mencari jarak terdekat di bagian kiri dan kanan secara rekursif
154         jarak1, koordinat1 = divideAndConquer(left_area, n, dimensi)
155         hitungEuclideanDC += 1
156         jarak2, koordinat2 = divideAndConquer(right_area, n, dimensi)
157         hitungEuclideanDC += 1
158

```

```

159 # Mencari koordinat dua titik terdekat
160 if jarak1 <= jarak2:
161     jarak = jarak1
162     tempNearest = koordinat1
163 elif jarak1 > jarak2:
164     jarak = jarak2
165     tempNearest = koordinat2
166
167 # Mencari jarak terdekat di antara kiri dan kanan jika jaraknya kurang dari strip (cek kondisi di tengah garis batas)
168 strip = []
169 for i in range(n):
170     if abs(listKoordinat[i][0] - listKoordinat[middle][0]) < jarak:
171         strip.append(listKoordinat[i])
172
173 # Mencari jarak terdekat di antara titik-titik yang ada di dalam strip
174 if dimensi == 1:
175     for i in range(len(strip)):
176         for j in range(i+1, len(strip)):
177             if (strip[j][0] - strip[i][0]) >= jarak and (strip[j][1] - strip[i][1]) >= jarak:
178                 break
179             newJarak = math.sqrt((pow((strip[j][0]-strip[i][0]),2))
180                                 + pow((strip[j][1]-strip[i][1]),2))
181             if newJarak < jarak:
182                 jarak = newJarak
183                 tempNearest = [strip[i], strip[j]]
184                 hitungEuclideanDC += 1
185     return jarak, tempNearest
186
187 elif dimensi == 2 or dimensi == 3:
188     # Sorting strip berdasarkan koordinat y menaik
189     strip_sorted = sortingY(strip)
190
191 # Mencari jarak terdekat di antara titik-titik yang ada di dalam strip
192 for i in range(len(strip_sorted)):
193     for j in range(i+1, len(strip_sorted)):
194         if (strip_sorted[j][0] - strip_sorted[i][0]) >= jarak and (strip_sorted[j][1] - strip_sorted[i][1]) >= jarak:
195             break
196         if dimensi == 2:
197             newJarak = math.sqrt((pow((strip_sorted[j][0]-strip_sorted[i][0]),2))+
198                                 (pow((strip_sorted[j][1]-strip_sorted[i][1]),2)))
199         elif dimensi == 3:
200             newJarak = math.sqrt((pow((strip_sorted[j][0]-strip_sorted[i][0]),2))+
201                                 (pow((strip_sorted[j][1]-strip_sorted[i][1]),2))+
202                                 (pow((strip_sorted[j][2]-strip_sorted[i][2]),2)))
203         if newJarak < jarak:
204             jarak = newJarak
205             tempNearest = [strip_sorted[i], strip_sorted[j]]
206             hitungEuclideanDC += 1
207     return jarak, tempNearest
208
209 # Menampilkan spesifikasi laptop untuk running program
210 def speklaptop():
211     print("=====")
212     print("SPESIFIKASI LAPTOP UNTUK RUNNING PROGRAM")
213     print("-----")
214     print("Processor:", platform.processor())
215     print("Operating System:", platform.platform())
216     print("Machine:", platform.machine())
217     total_ram = psutil.virtual_memory().total / (1024.0 **3)
218     print("RAM:",total_ram, "GB")
219     print("=====")

```

e) Main.py

Program ini merupakan program utama yang berfungsi untuk menjalankan program-program yang telah dibuat pada file lainnya. Dalam program main ini, terdapat satu fungsi yaitu `def mainSolvingProblem` yang menerima dimensi dan koordinat. Fungsi ini bertujuan untuk menjalankan fungsi-fungsi yang telah dibuat pada *file* lain sebelumnya. Fungsi-fungsi lain yang diimplementasikan dalam fungsi di *main* ini adalah sebagai berikut:

- 1) Fungsi untuk menampilkan proyeksi titik-titik koordinat, semua titik memiliki warna merah
- 2) Fungsi untuk menampilkan spesifikasi laptop yang digunakan untuk running program

- 3) Fungsi untuk melakukan pencarian dua titik koordinat terdekat dengan algoritma *brute force* dan algoritma *divide and conquer* serta menampilkan hasil jarak terdekat, berapa kali perhitungan jarak *Euclidean*, dan waktu yang diperlukan untuk *running* program
- 4) Fungsi untuk menampilkan proyeksi kedua titik yang memiliki jarak terdekat, untuk titik yang jaraknya terdekat akan berwarna biru, dan titik-titik lainnya akan berwarna merah

Untuk urutan proses main programnya adalah program akan menampilkan *splash screen* pembuka dan menu untuk menanyakan kepada *user* mengenai input jumlah titik dan juga memilih dimensi (pilihan antara 1D, 2D, atau 3D). Setelah itu, program akan menginput koordinat titik secara random dengan pembangkit bilangan acak. Jumlah titik yang diinput adalah sejumlah titik yang dimasukkan *user* sebelumnya. Setelah itu, program akan melakukan *sorting* titik secara terurut berdasarkan nilai absis pada koordinat titik. Setelahnya, program akan menjalankan fungsi `def mainSolvingProblem` dengan proses-proses yang sudah dijelaskan sebelumnya. Jika fungsi ini sudah selesai dijalankan, program akan selesai dan menampilkan *splash screen* penutup. Untuk programnya sendiri adalah sebagai berikut:

```
1  import inputting as inp
2  import processing as process
3  import projection as project
4  import splash as splash
5  import time
6
7  # Fungsi utama untuk mencari jarak terdekat antar titik dan menampilkan hasilnya
8  def mainSolvingProblem(dimensi, koordinatSorting):
9      project.projectionBefore(koordinatSorting, dimensi)
10
11      print("")
12
13      process.spekLaptop()
14
15      print("")
16
17      # Melakukan proses brute force
18      start_bf_time = time.time()
19      jarak = process.bruteForce(koordinatSorting, dimensi)
20      stop_bf_time = time.time()
21
22      # Menampilkan hasil dengan Algoritma Brute Force
23      print("=====")
24      print("MENCARI DENGAN ALGORITMA BRUTE FORCE")
25      print("-----")
26      print("Jarak terdekat antar titik:", jarak)
27      print("Banyaknya perhitungan operasi Euclidean:", process.hitungEuclideanBF)
28      print("Waktu menjalankan program: %.7s detik" % (stop_bf_time - start_bf_time))
29      print("=====")
30
31      print("")
32
```

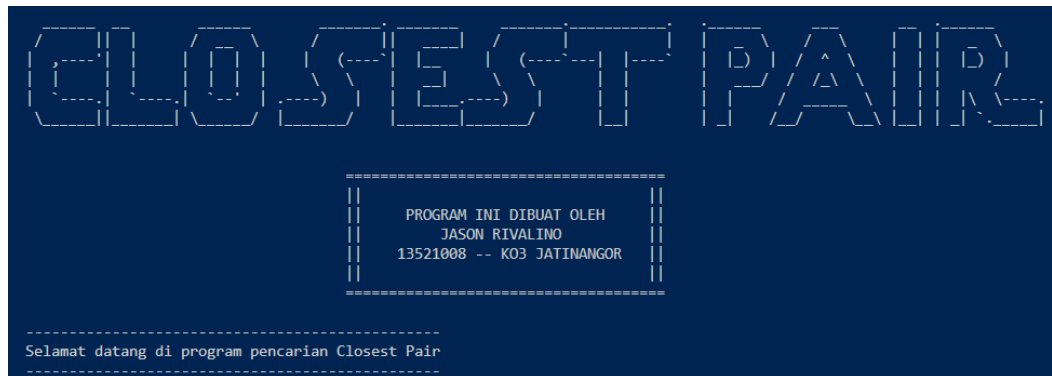
```
33     # Melakukan proses divide and conquer
34     start_bf_time = time.time()
35     jarak, koordinat = process.divideAndConquer(koordinatSorting, len(koordinatSorting), dimensi)
36     stop_bf_time = time.time()
37
38     # Menampilkan hasil dengan Algoritma Divide and Conquer
39     print("=====")
40     print("MENCARI DENGAN ALGORITMA DIVIDE AND CONQUER")
41     print("-----")
42     print("Jarak terdekat antar titik:", jarak)
43     print("Banyaknya perhitungan operasi Euclidean:", process.hitungEuclideanDC)
44     print("Waktu menjalankan program: %.7s detik" % (stop_bf_time - start_bf_time))
45     print("=====")
46
47     print("")
48
49     # Menampilkan koordinat titik terdekat dan proyeksinya
50     print("=====")
51     print("Koordinat titik terdekat: ", koordinat)
52     project.projectionAfter(koordinatSorting, koordinat, dimensi)
53
54
55 # MAIN PROGRAM
56
57 # Opening Splash
58 splash.openingSplash()
59
60 # Inisialisasi variabel
61 process.hitungEuclideanBF = 0
62 process.hitungEuclideanDC = 0
63 n = 0
64
65 # Input jumlah titik
66 n = inp.inputTitik(n)
67
68 # Input dimensi
69 print("")
70 dimensi = inp.inputDimension(n)
71
72 # Input koordinat
73 listKoordinat = inp.inputKoordinat(n, dimensi)
74
75 # Melakukan proses sorting
76 koordinatSorting = process.sortingX(listKoordinat)
77
78 # Memanggil fungsi utama untuk mencari jarak terdekat antar titik dan menampilkan hasilnya
79 mainSolvingProblem(dimensi, koordinatSorting)
80
81 print("")
82
83 # Closing Splash
84 splash.closingSplash()
```

BAB IV

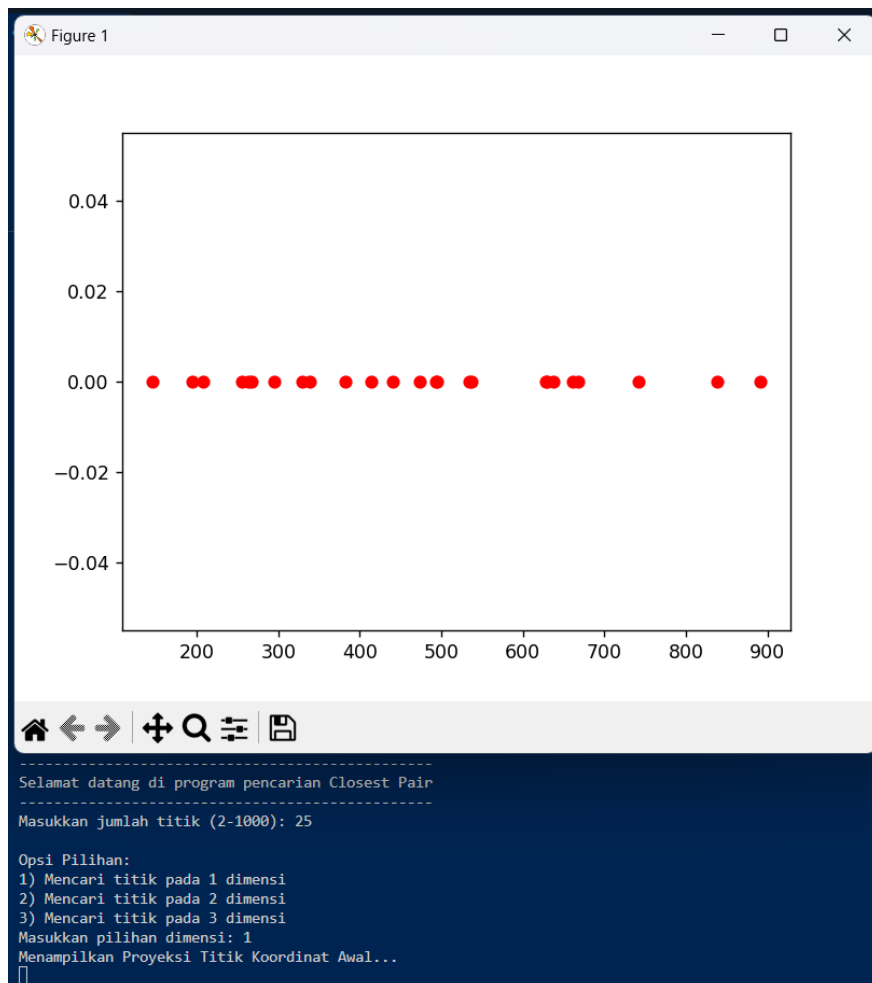
EKSPERIMEN

I. Hasil Running Program

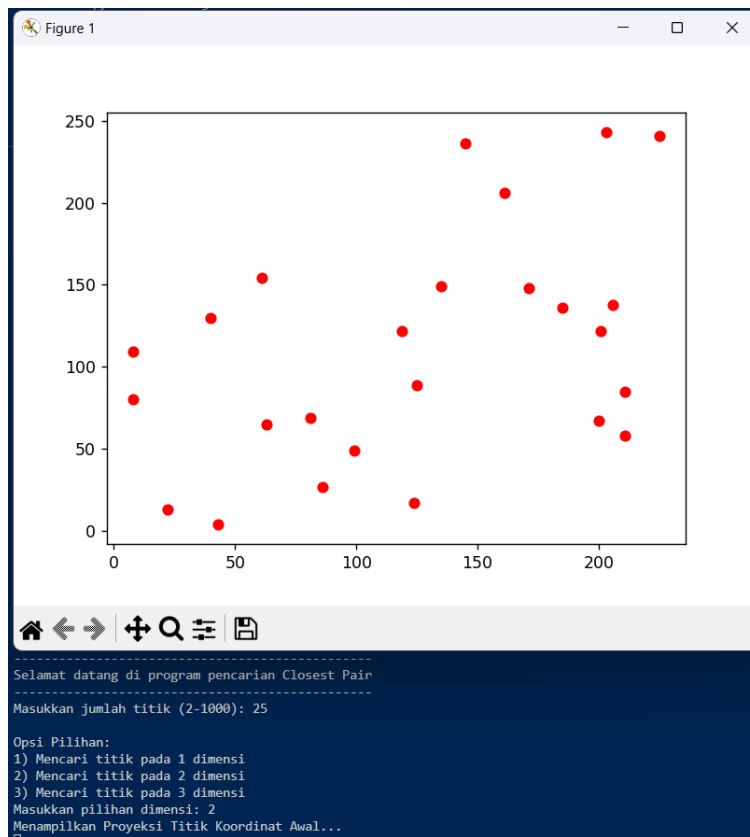
a) Splash Screen Pembuka



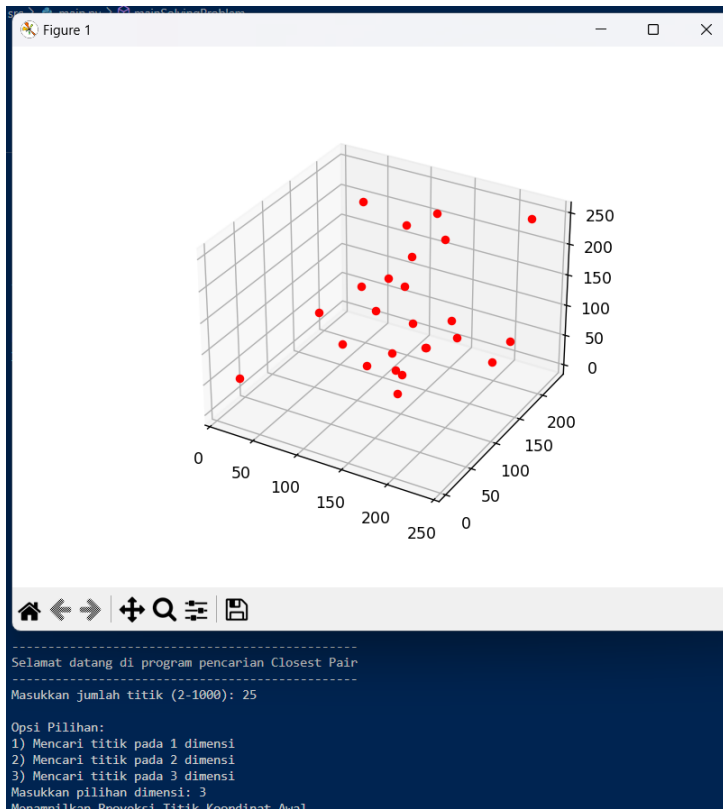
b) Memasukkan jumlah titik dan ukuran dimensi serta memproyeksikan titik-titik koordinat dalam bidang satu dimensi



- c) Memasukkan jumlah titik dan ukuran dimensi serta memproyeksikan titik-titik koordinat dalam bidang dua dimensi



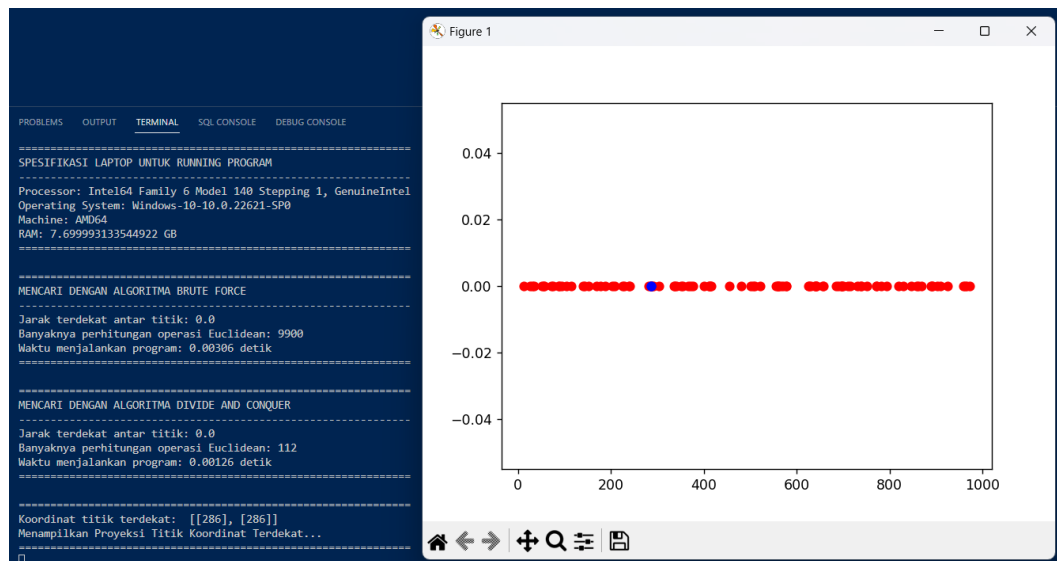
- d) Memasukkan jumlah titik dan ukuran dimensi serta memproyeksikan titik-titik koordinat dalam bidang tiga dimensi



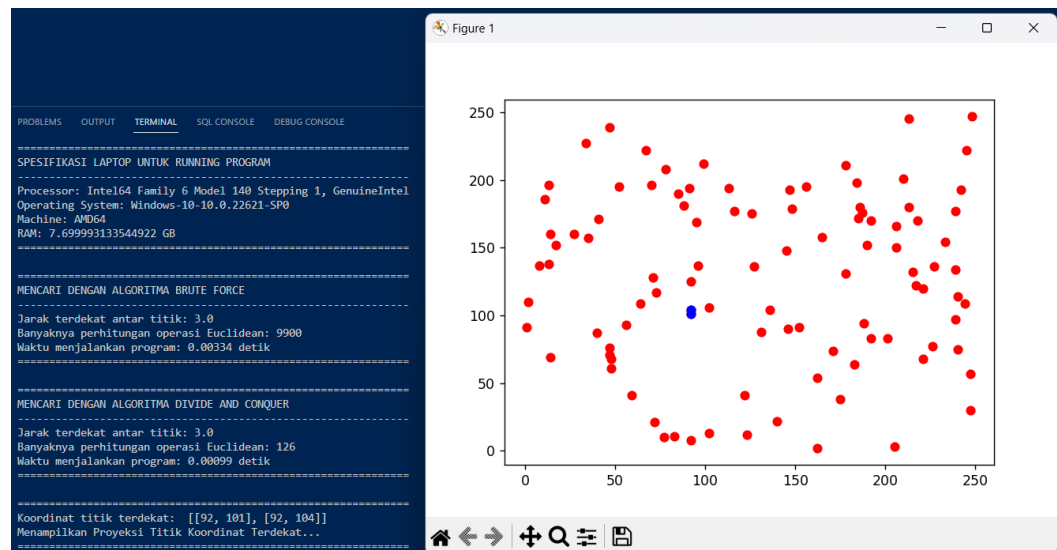
IF2211

Strategi Algoritma

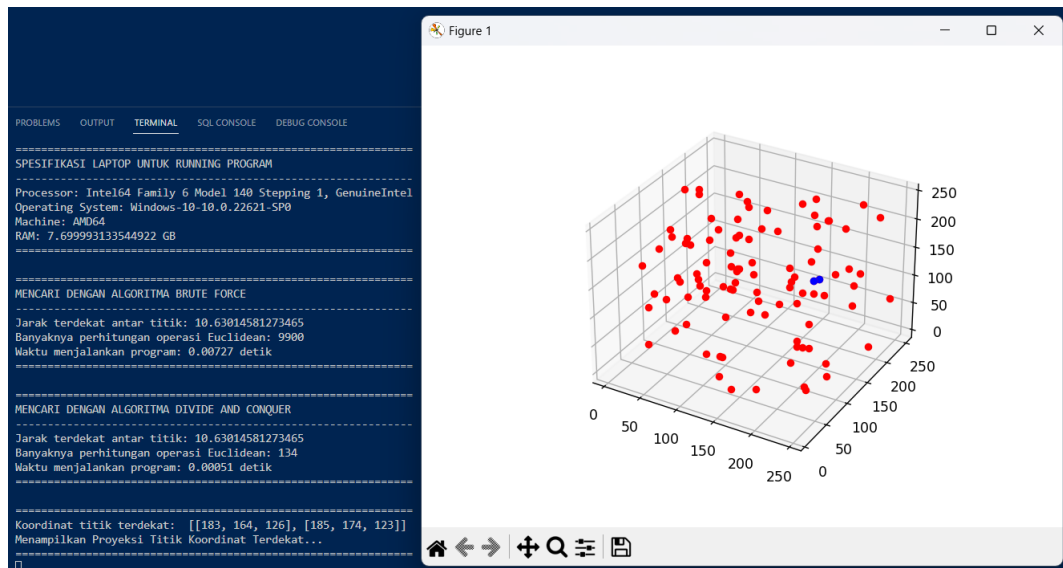
- e) Menampilkan spesifikasi laptop untuk *running* program, hasil jarak terdekat dengan algoritma *brute force* dan algoritma *divide and conquer*, banyaknya perhitungan *euclidean*, waktu menjalankan program, koordinat titik terdekat, dan juga proyeksi untuk titik koordinat terdekat dalam bidang satu dimensi



- f) Menampilkan spesifikasi laptop untuk *running* program, hasil jarak terdekat dengan algoritma *brute force* dan algoritma *divide and conquer*, banyaknya perhitungan *euclidean*, waktu menjalankan program, koordinat titik terdekat, dan juga proyeksi untuk titik koordinat terdekat dalam bidang dua dimensi



- g) Menampilkan spesifikasi laptop untuk *running* program, hasil jarak terdekat dengan algoritma *brute force* dan algoritma *divide and conquer*, banyaknya perhitungan *euclidean*, waktu menjalankan program, koordinat titik terdekat, dan juga proyeksi untuk titik koordinat terdekat dalam bidang tiga dimensi

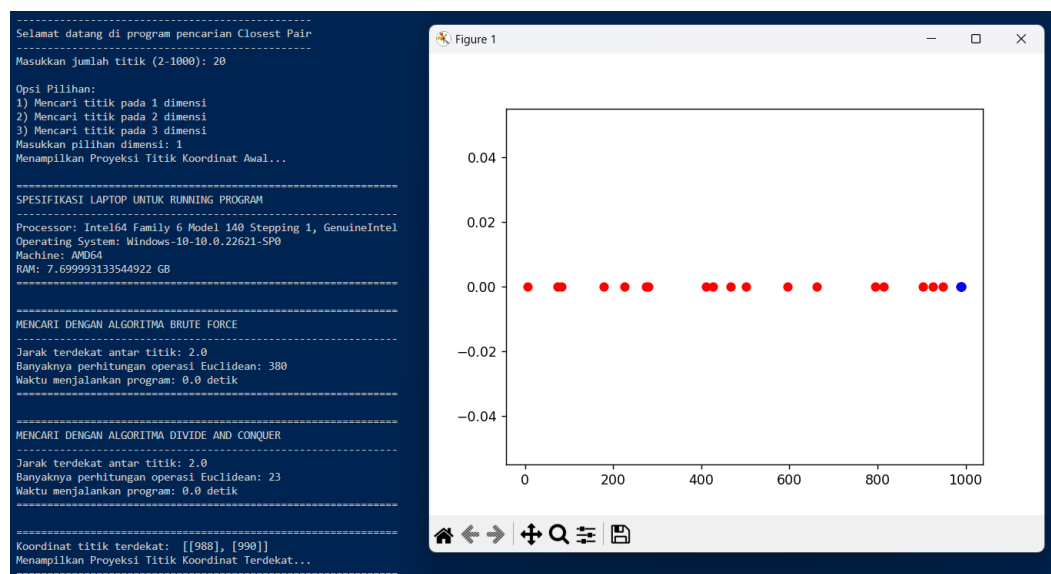


- h) Menampilkan splash screen closing jika program sudah selesai

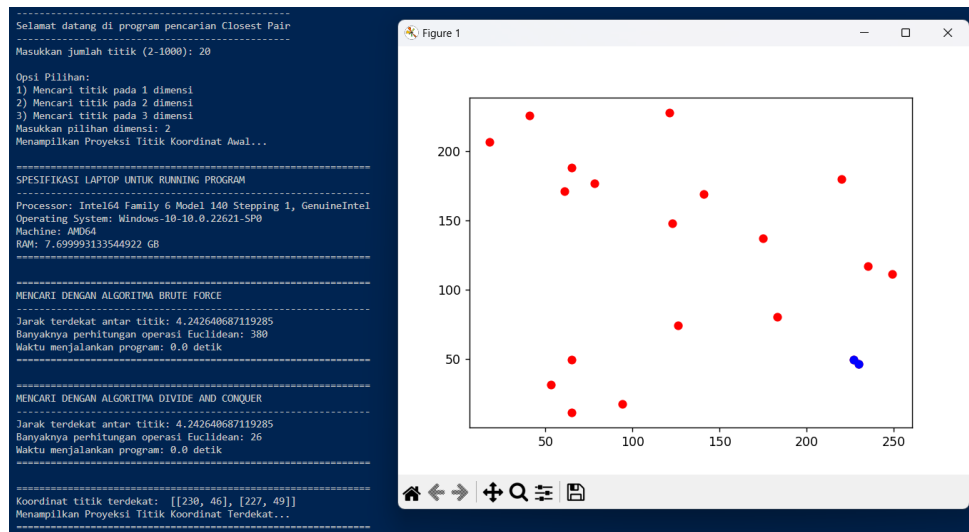


II. Uji Coba Hasil Program

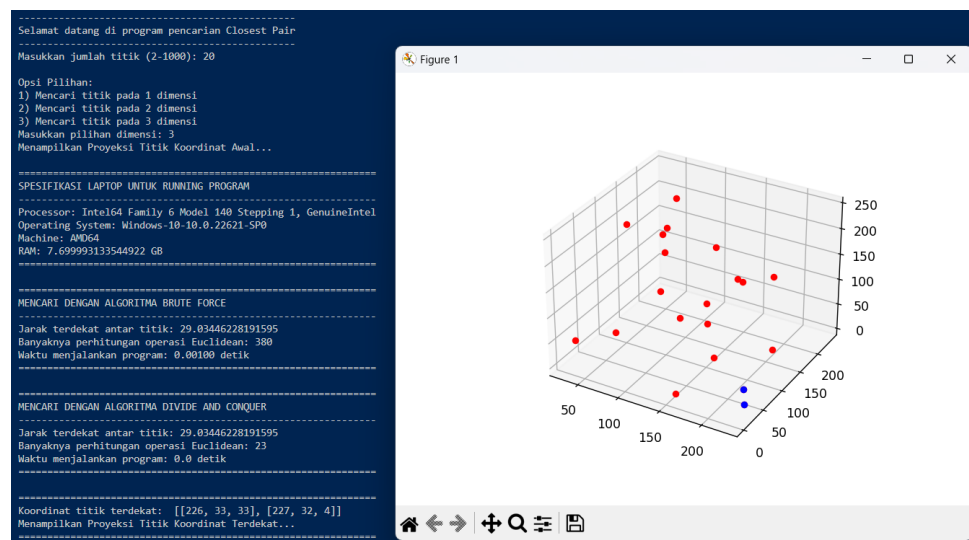
- a) Pengujian 20 titik
1) Bidang 1 dimensi



2) Bidang 2 dimensi

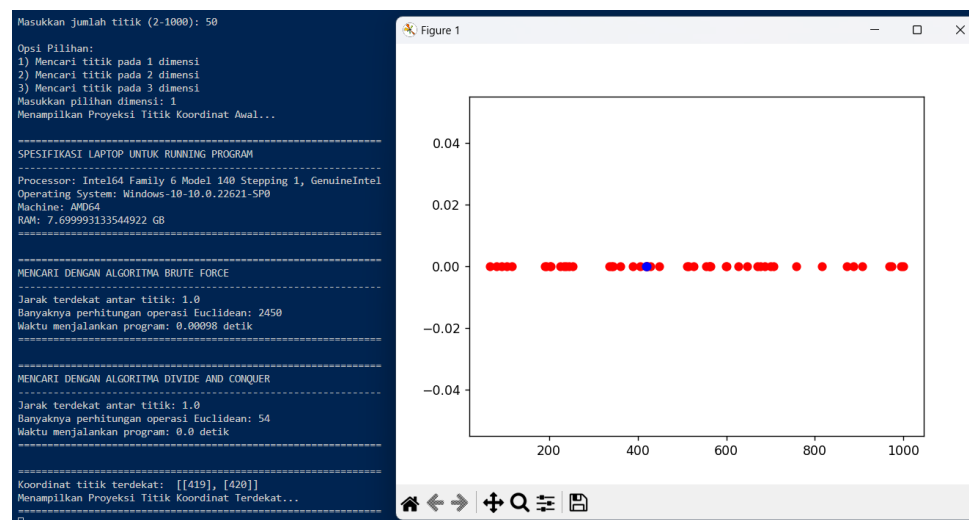


3) Bidang 3 dimensi

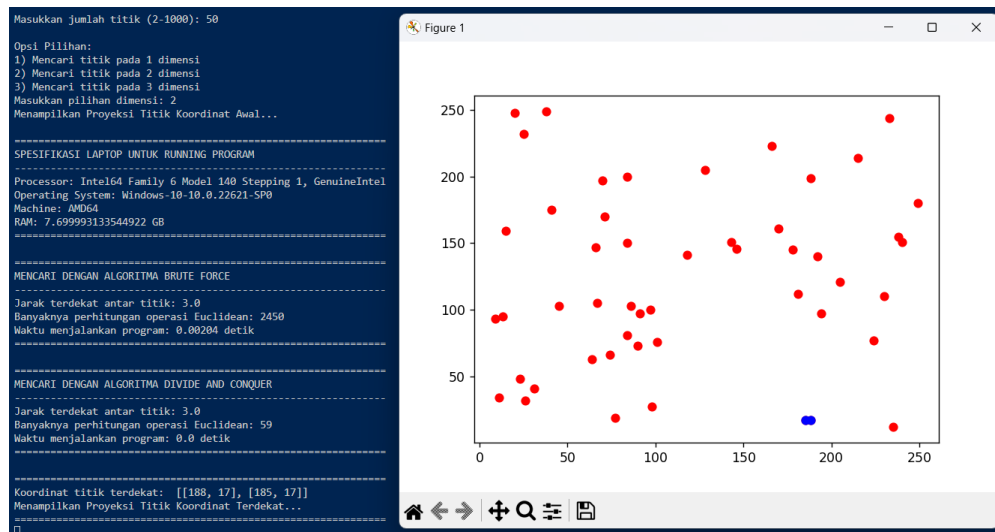


b) Pengujian 50 titik

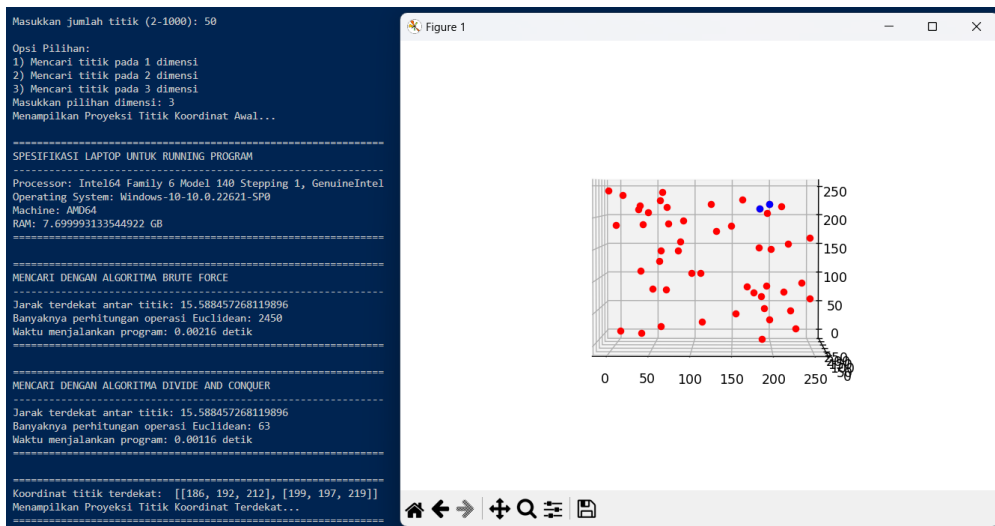
1) Bidang 1 dimensi



2) Bidang 2 dimensi

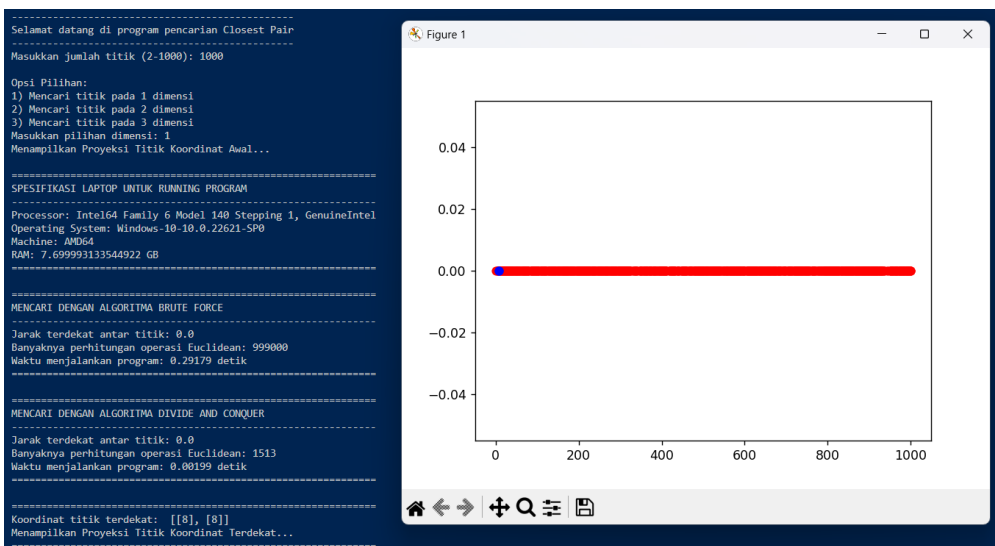


3) Bidang 3 dimensi

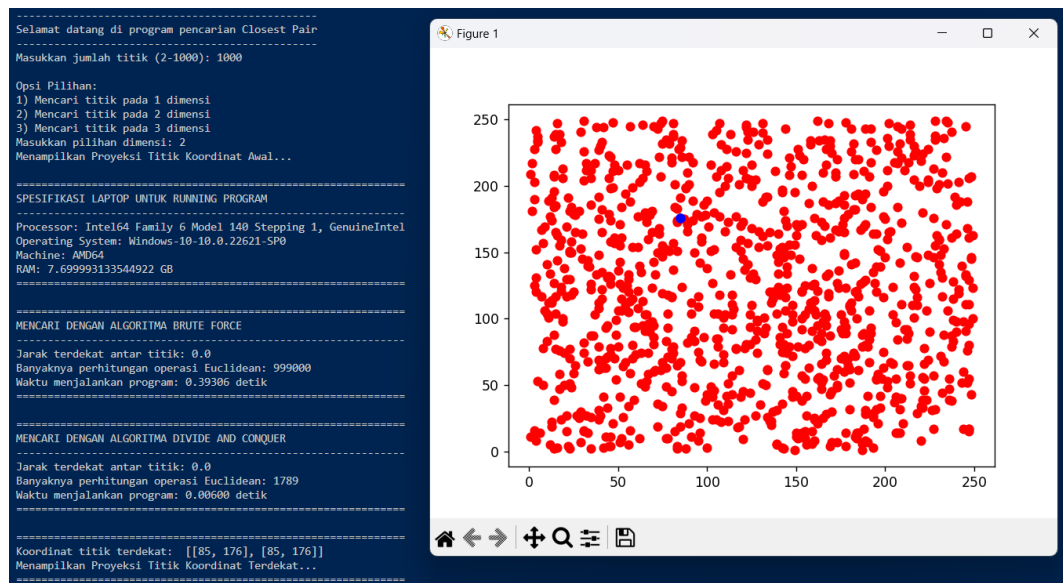


c) Pengujian 1000 titik

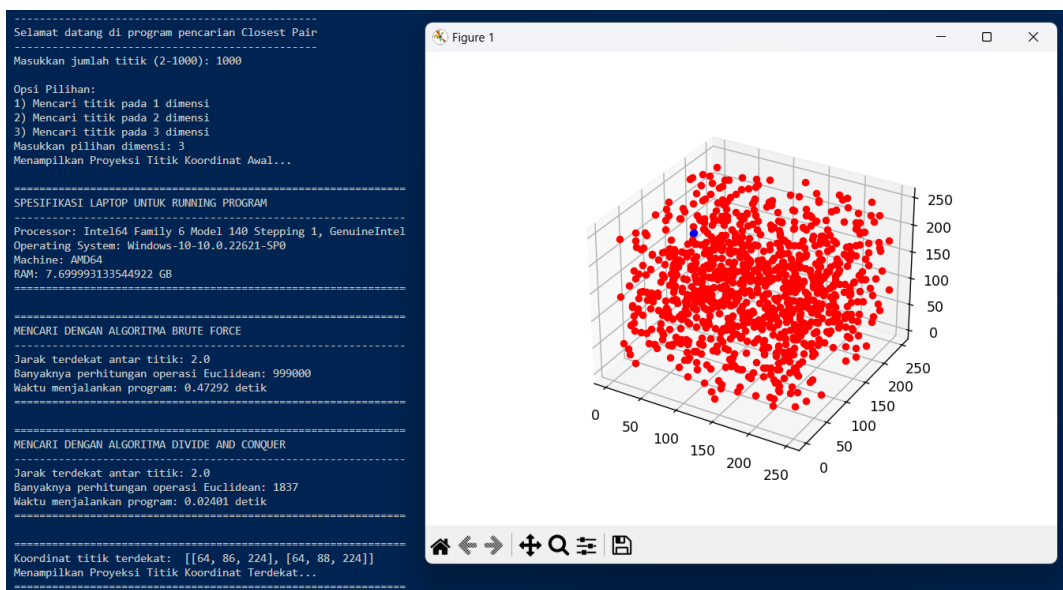
1) Bidang 1 dimensi



2) Bidang 2 dimensi



3) Bidang 3 dimensi



BAB V

KESIMPULAN

Kesimpulan yang didapat dari pengerjaan tugas kecil ini adalah sebagai berikut:

- 1) Algoritma *Divide and Conquer* dapat dipergunakan untuk menyelesaikan beberapa macam persoalan dan memiliki kompleksitas algoritma yang lebih mangkus jika dibandingkan dengan algoritma *Brute Force*
- 2) Persoalan pencarian titik terdekat (*closest pair*) dapat menggunakan algoritma *Divide and Conquer* untuk mencari solusi dari permasalahannya

REFERENSI

[1] R. Munir (2022). Algoritma Divide and Conquer Bagian 1 [Powerpoint Slides]. Available: [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)-Bagian1.pdf)

LAMPIRAN

Pranala Github: https://github.com/jasonrivalino/Tucil2_13521008

Checklist:

Poin	Judul Fitur	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil <i>running</i>	✓	
3	Program dapat membaca input / generate sendiri dan memberikan luaran	✓	
4	Luaran program sudah benar (solusi closest pair benar)	✓	
5	Bonus 1 dikerjakan	✓	
6	Bonus 2 dikerjakan	✓	

Note: untuk bonus 2, implementasi yang dibuat hanya bisa untuk mencari sepasang titik terdekat dalam bidang 1-3 dimensi