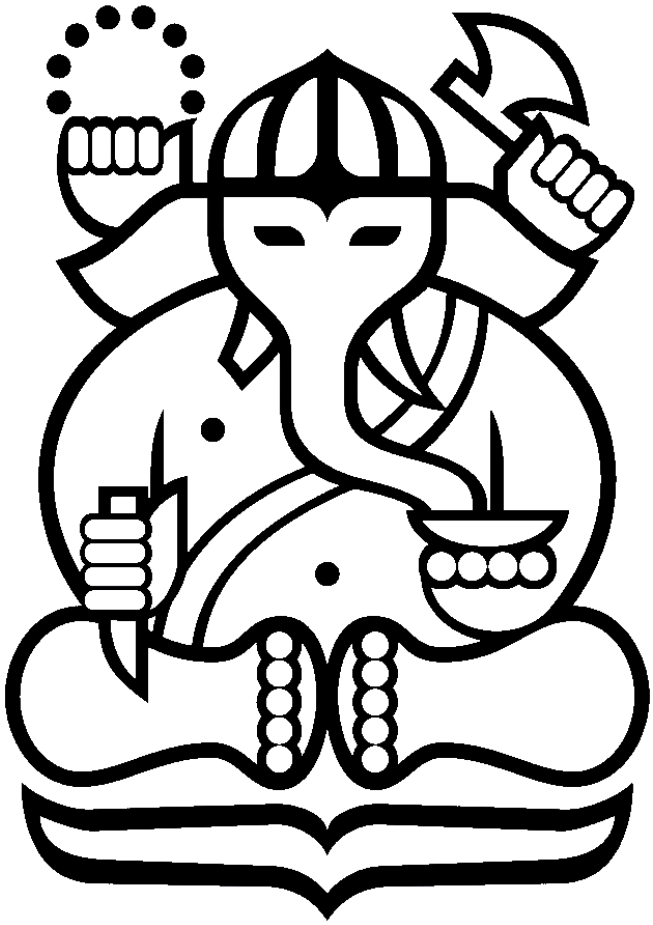
Tugas Kecil 2 IF2211 Strategi Algoritma

Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer



Disusun oleh :

13521019 - Ditra Rizqa Amadia

13521024 - Ahmad Nadil

INSTITUT TEKNOLOGI BANDUNG

2023

Daftar Isi

BAB I

DESKRIPSI MASALAH DAN ALGORITMA

1.1 Algoritma Divide and Conquer

Algoritma *divide and conquer* merupakan sebuah pendekatan dalam pemecahan masalah. Pada algoritma *divide and conquer*, terdapat dua elemen utama, yaitu *Divide and Conquer*. Untuk menyelesaikan permasalahan dengan menggunakan *divide and conquer*, pertama-tama permasalahan dapat dibagi menjadi beberapa upa-persoalan yang memiliki kemiripan persoalan semula namun berukuran lebih kecil terlebih dahulu. Idealnya, masing-masing upa-persoalan memiliki ukuran yang sama. Setelah membagi-bagi permasalahan menjadi bagian-bagian kecil, selesaikan masing-masing permasalahn (*conquer*) secara langsung atau secara rekursif apabila persoalan masih berukuran besar. Terakhir, gabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula. Dikarenakan setiap upa-persoalan memiliki karakteristik yang sama, algoritma *Divide and Conquer* seringkali lebih natural diungkapkan dalam skema rekursif dan diimplementasikan dengan fungsi rekursif pula.

1.2 Pencarian Pasangan TItik Terdekat

Persoalan pencarian pasangan titik terdekat adalah sebuah masalah analisis data dan geometrik dalam komputasi dalam mencari titik terdekat dalam sebuah set titik terhadap titik lainnya., Secara sederhana, persoalan ini dapat diartikan sebagai mencari titik paling dekat dari titik lain dalam satu set titik.

Persoalan pencarian titik terdekat dirumuskan dalam sebuah bidang, bisa dalam bidang datar (2D), bidang ruang (3D), ataupun dalam ruang vektor R^n. Contoh dari persoalan ini adalah terdapat *n* buah titik pada ruang 3D, setiap titik *P* di dalam ruang dinyatakan dengan koordinat *P = (x,y,z)*, dari sana, carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titik *P1 = (x1,y1,z1)* dan *P2 = (x2,y2,z2)* dapat dihitung dengan menggunakan rumus Euclidean sebagai berikut :

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2}$$

Persoalan ini dapat diterapkan dalam berbagai aplikasi, seperti dalam pengenalan pola, pengolahan citra, pengelompokan data, dan sebagainya. Contoh penggunaannya adalah dalam klasifikasi gambar, di mana sebuah gambar baru diklasifikasikan dengan mencari gambar-gambar yang paling mirip dengannya di dalam sebuah dataset, atau dalam optimasi rute, di mana titik terdekat digunakan untuk menentukan rute tercepat antara dua lokasi.

1.3 Algoritma Penyelesaian Permasalahan Pasangan Titik Terdekat dengan Pendekatan Divide and Conquer

Dalam menyelesaikan permasalahan pasangan titik terdekat secara algoritmik, pengguna akan menggunakan pendekatan *Divide and Conquer*. Adapun langkah-langkah penyelesaian permasalahan dalam algoritma dapat dijelaskan secara deskriptif sebagai berikut :

1. Program meminta pengguna untuk melakukan input dua buah integer, yaitu *n* (jumlah titik yang akan dibangkitkan secara acak) dan *d* (dimensi dari titik yang akan dibangkitkan).
2. Sebagai pembanding hasil algoritma *divide and conquer* nanti, program akan melakukan pencarian jarak titik terdekat dengan metode *brute force* terlebih dahulu, dengan metode setiap titik akan dihitung jaraknya dengan semua titik lainnya dan akan dicari jarak terdekat.
3. Urutkan secara menaik titik-titik yang akan dilakukan pengecekan berdasarkan abisnya (sumbu x). Pada program ini, penulis menggunakan salah satu metode *Sorting* yang memanfaatkan algoritma *Divide and Conquer*, yaitu *Quicksort*.
4. Dalam metode *Quicksort* yang diimplementasikan, dipilih sebuah elemen dari data yang disebut pivot, pada program ini pengguna menggunakan elemen pertama sebagai pivot. Lalu, lakukan proses *partitioning* dimana data dibagi menjadi dua bagian, di mana setiap elemen di bagian pertama lebih kecil pivot dan setiap elemen di bagian kedua lebih besar dari pivot. Setelah itu, lakukan rekursif dalam pengurutan pada setiap bagian yang terbentuk pada langkah sebelumnya. Akhirnya, gabungkan kembali setiap bagian yang sudah terurut pada langkah sebelumnya.

5. Titik-titik yang telah diurutkan lalu akan dihitung jarak terdekatnya menggunakan algoritma *Divide and Conquer*. Terdapat beberapa tahap untuk mendapatkan pasangan titik terdekat, yaitu *Solve, Divide, Conquer,* dan *Combine*.
6. *Solve*, yaitu jika *n* (banyak titik) adalah 2, maka akan dihitung jaraknya langsung menggunakan rumus *Euclidean*. Rumus Euclidean yang dibuat oleh penulis akan mengiterasi sebesar panjang dari titik tersebut, sehingga dapat menghitung jarak dari titik-titik yang berada pada dimensi berbeda (R^n)
7. *Divide*, bagi himpunan titik ke dalam dua bagian, S1 dan S2, setiap bagian mempunyai jumlah titik yang sama. S1 dan S2 merupakan sub-himpunan dari himpunan titik awal yang memiliki titik sejumlah *n*/2.
8. *Conquer*, secara rekursif, S1 dan S2 akan dihitung kembali jarak titik-titik terdekatnya dan akan dilakukan pembagian sub-himpunan secara terus menerus sampai mencapai basis, ketika array titik bersisa dua buah titik, agar dapat dihitung jarak dari dua titik tersebut. Dari S1 dan S2 akan dibandingkan, manakah yang memiliki jarak terdekat, lalu kemudian akan dijadikan pembanding perhitungan pada titik yang terdapat pada *Strip* (dua titik yang terpisahkan oleh garis maya sebesar jarak terdekat dari S1 atau S2. Pada *Strip* ini akan dilakukan perhitungan jarak titik terdekatnya.
9. *Combine*, terdapat tiga kemungkinan letak titik tersebut, yaitu dalam bagian S1, bagian S2, atau dalam *Strip*. Nilai jarak yang didapat dari ketiga perhitungan akan dibandingkan dan dilihat manakah yang memiliki jarak terdekat.
10. Setelah mendapatkan titik terdekat serta jaraknya, akan dibandingkan solusinya dengan solusi yang didapat dengan algoritma *brute force* lalu akan dilakukan visualisasi dari titik-titik tersebut (visualisasi hanya akan dilakukan pada titik berdimensi 2 dan 3).

BAB II

IMPLEMENTASI ALGORITMA DALAM BAHASA PYTHON

Dalam pembuatan program ini, penulis menggunakan bahasa pemrograman Python. Struktur dari program ini terbagi menjadi 6 file, yaitu ***main.py*, Splash.py *Euclidean.py, Sort.py, ClosestPair.py, System.py,*** dan ***Plot.py.***

2.1 File main.py

File ini merupakan *driver* utama dari program ini, sehingga tidak terdapat fungsi di dalamnya, hanya berisi menu utama dari program ini serta deklarasi variabel yang akan digunakan.

Variable Name	Description
`int n`	Variabel banyaknya titik pada permasalahan
`int d`	Variabel ruang dimensi pada permasalahan
`array points`	Variabel yang menyimpan array dari titik
`float start`	Variabel yang menyimpan waktu mulai suatu algoritma
`float end`	Variabel yang menyimpan waktu akhir suatu algoritma
`tuple min`	Variabel tuple yang menyimpan hasil algoritma *brute force* berupa 2 titik dengan jarak terdekat beserta jaraknya
`tuple closest_pair`	Variabel tuple yang menyimpan hasil algoritma *divide and conquer*
`float bf_extime`	Variabel yang menyimpan durasi eksekusi algoritma *brute force*
`int bg_euclidop`	Variabel yang menyimpan banyaknya operasi *euclidean distance* yang digunakan dalam algoritma *brute force*
`float dc_extime`	Variabel yang menyimpan durasi eksekusi algoritma

	divide and conquer
`int dc_euclidop`	Variabel yang menyimpan banyaknya operasi *euclidean distance* yang digunakan dalam algoritma *divide and conquer*

2.2 File Splash.py

File ini berisi fungsi untuk menampilkan *splash screen* pada layar *user*.

Methods	Description
`Void displaySplash()`	Menampilkan *splash screen* pada layar

2.3 File Euclidian.py

File ini berisi fungsi rumus euclidean untuk menghitung jarak dua titik.

Methods	Description
`float EuclideanDistance (point 1, point 2)`	Menerima 2 buah titik dan mengembalikan jarak antara 2 titik tersebut

2.4 File Sort.py

File ini berisi fungsi untuk melakukan pengurutan elemen-elemen titik.

Methods	Description
`array SortPointsByX (points)`	Menerima sebuah array yang berisi titik-titik dan akan dilakukan pengurutan berdasarkan absisnya (sumbu X).
`array SortPointsByY (points)`	Menerima sebuah array yang berisi titik-titik dan akan dilakukan pengurutan berdasarkan oordinatnya (Sumbu Y).

2.5 File ClosestPair.py

File ini berisi fungsi-fungsi untuk mencari solusi dari permasalahan *The Closest-Pair Problem.*

Methods	Description
`tuple closest_pair(points)`	Menerima *array of points* dan mengembalikan tuple berupa titik 1, titik 2, dan jarak antara kedua titik tersebut dengan jarak merupakan jarak terdekat dari kumpulan titik-titik yang ada
`tuple closest_pair_strip(strip, d)`	Menerima *array of points strip* dan *integer* d. *Array of points* berisikan titik dengan koordinat x di mana -d < x < d. Mengembalikan tuple berupa titik 1, titik 2, dan jarak antar kedua titik tersebut
`tuple brute_force(points)`	Menerima *array of points* dan mengembalikan titik 1, titik 2, dan jarak antara kedua titik tersebut di mana jarak merupakan jarak terkecil dari kumpulan titik-titik yang ada

2.6 File System.py

File ini berisi fungsi untuk menampilkan spesifikasi *hardware* yang digunakan oleh *user*.

Methods	Description
`void displaySpecification()`	Menampilkan spesifikasi *hardware* yang digunakan oleh user

2.7 File Plot.py

File ini berisi fungsi untuk menampilkan visualisasi solusi dari permasalahan kepada *user*.

Methods	Description

`void result_plot(points, closest_pair, dimension)`	Menerima *array of points, tuple* sepasang titik dengan jarak terdekat, dan ruang dimensi. Menampilkan visualisasi solusi permasalah kepada *user*

2.8 Library

Terdapat juga beberapa library yang digunakan untuk program ini, antara lain :

- `numpy`
- `time`
- `matplotlib.pyplot`
- `platform`
- `psutil`
- `math`

BAB III

SOURCE CODE PROGRAM

3.1 Repository Program

Repository program dapat diakses melalui tautan *GitHub* berikut : https://github.com/IceTeaXXD/Tucil2_13521019_13521024

3.2 Source Code Program

3.2.1 main.py

```
import numpy as np
from Splash import *
from System import *
from ClosestPair import*
from Plot import*
from time import time
from Sort import*
import Euclidean as e

# Splash Screen
displaySplash()

# Inputs
n = int(input("\nMasukkan banyaknya titik: "))
while (n < 2):
    print("Banyaknya titik harus lebih dari 1!")
    n = int(input("Masukkan banyaknya titik: "))
d = int(input("Masukkan dimensi: "))
while (d < 2):
    print("Dimensi harus lebih dari 1!")
    d = int(input("Masukkan dimensi: "))
points = np.around(np.random.uniform(-1000, 1000, (n, d)),decimals=3)

# Hardware Specification
```

```
print("\n== HARDWARE SPECIFICATION ==========----------")
displaySpecification()

# Solution by Brute Force
print("\n== BRUTE FORCE ==========----------")
start = time()
min = brute_force(points)
end = time()
bf_extime = (end-start)
bf_euclidop = e.euclidCounter
print('Closest pair of points:')
print(f'Point 1 : \t\t{min[0]}')
print(f'Point 2 : \t\t{min[1]}')
print(f'Distance: \t\t{min[2]}')
print(f'Execution time: \t{"{:.4f}".format(bf_extime)} s')
print(f'Euclidean Distance Operations: {bf_euclidop}')

# Solution by Divide and Conquer
print("\n== DIVIDE AND CONQUER ==========----------")
start = time()
points = SortPointsByX(points)
closest_pair = closest_pair(points)
end = time()
dc_extime = (end-start)
dc_euclidop = e.euclidCounter-bf_euclidop
print('Closest pair of points:')
print(f'Point 1 : \t\t{closest_pair[0]}')
print(f'Point 2 : \t\t{closest_pair[1]}')
print(f'Distance: \t\t{closest_pair[2]}')
print(f'Execution time: \t{"{:.4f}".format(dc_extime)} s')
print(f'Euclidean Distance Operations: {dc_euclidop}')


# Comparison
print("\n== SOLUTIONS COMPARISON ==========----------")
if(closest_pair[2] == min[2]):
    print("Solutions match")
else:
```

```
    print("Solutions does not match")
if(dc_extime < bf_extime):
    print(f"Divide and conquer is faster by
{'{:.4f}'.format(bf_extime-dc_extime)} s")
else:
    print(f"Brute force is faster by {'{:.4f}'.format(dc_extime-bf_extime)} s")
if(dc_euclidop < bf_euclidop):
    print(f"Divide and conquer used {bf_euclidop-dc_euclidop} less euclidean
distance operations")
else:
    print(f"Brute force used {dc_euclidop-bf_euclidop} less euclidean distance
operations")
print("\n")

result_plot(points, closest_pair, d)
```

3.2.2 Splash.py

```
def displaySplash():
    print("\nTHE CLOSEST-PAIR PROBLEM SOLVER")
    print("\n              by              ")
    print("\n Ditra Rizqa Amadia (13521019) ")
    print("     Ahmad Nadil (13521025)    ")
```

3.2.3 Euclidean.py

```
import math

euclidCounter = 0
def EuclideanDistance(point1, point2):
    # I.S. point1 and point2 are tuples of the same length
    # F.S. returns the Euclidean distance between point1 and point2
    global euclidCounter
    euclidCounter += 1
    return math.sqrt(sum([(point1[i] - point2[i]) ** 2 for i in
range(len(point1))])) # The formula counts the distance between two points in
n-dimensional space
```

3.2.4 Sort.py

```
def SortPointsByX(points):
    # I.S. points is a list of tuples
    # F.S. returns a list of tuples sorted by the first element (x coordinate) of
each tuple
    if len(points) <= 1:
        return points
    else:
        pivot = points[0]
        less = [point for point in points[1:] if point[0] < pivot[0]]
        greater = [point for point in points[1:] if point[0] >= pivot[0]]
        return SortPointsByX(less) + [pivot] + SortPointsByX(greater)

def SortPointsByY(points):
    # I.S. points is a list of tuples
    # F.S. returns a list of tuples sorted by the second element (y coordinate)
of each tuple
    if len(points) <= 1:
        return points
    else:
        pivot = points[0]
        less = [point for point in points[1:] if point[1] < pivot[1]]
        greater = [point for point in points[1:] if point[1] >= pivot[1]]
        return SortPointsByY(less) + [pivot] + SortPointsByY(greater)
```

3.2.5 ClosestPair.py

```
from Euclidean import*
from Sort import*

def closest_pair(points):
    # I.S. points is a list of tuples
    # F.S. returns the tuple consists of point1, point2, and distance
    n = len(points)
    if n <= 1: # Base case (error handling if there is only one point)
        return None, None, float('inf')
    elif n == 2: # Base case
        return points[0], points[1], EuclideanDistance(points[0], points[1])
```

```
    else: # Recursive case
        # Divide the points into two halves
        left_x = points[:n//2]
        right_x = points[n//2:]

        # Find the closest pair in each half
        left_min_pair = closest_pair(left_x)
        right_min_pair = closest_pair(right_x)
        min_pair = left_min_pair if left_min_pair[2] < right_min_pair[2] else
right_min_pair

        # Find the closest pair that crosses the midpoint (strip)
        mid_x = points[n//2][0]
        strip = []
        for point in points:
            if abs(point[0] - mid_x) < min_pair[2]:
                strip.append(point)

        strip_min_pair = closest_pair_strip(strip, min_pair[2])
        return strip_min_pair if strip_min_pair[2] < min_pair[2] else min_pair

def closest_pair_strip(strip, d):
    # I.S. strip is a list of tuples and d is the distance between the closest
pair
    # F.S. returns the tuple consists of point1, point2, and distance
    min_pair = None, None, d
    sorted_y = SortPointsByY(strip)
    n = len(sorted_y)
    # Compare each point with all the points after it (brute force)
    for i in range(n):
        for j in range(i+1, n):
            if abs(sorted_y[j][1] - sorted_y[i][1]) >= d:
                break
            if EuclideanDistance(sorted_y[i], sorted_y[j]) < min_pair[2]:
                min_pair = sorted_y[i], sorted_y[j],
EuclideanDistance(sorted_y[i], sorted_y[j])
    return min_pair
```

```
def brute_force(points):
    # I.S. points is a list of tuples
    # F.S. returns the tuple consists of point1, point2, and distance
    min = None, None, 0
    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            if min[2] == 0: # If min is not initialized
                min = points[i], points[j], EuclideanDistance(points[i],
points[j])
            else:
                if min[2] > EuclideanDistance(points[i], points[j]):
                    min = points[i], points[j], EuclideanDistance(points[i],
points[j])
    return min
```

3.2.6 System.py

```
import platform, psutil

my_sys = platform.uname()
cpufreq = psutil.cpu_freq()

def displaySpecification():
    print(f"System: \t{my_sys.system}")
    print(f"Node name: \t{my_sys.node}")
    print(f"Version: \t{my_sys.version}")
    print(f"Machine: \t{my_sys.machine}")
    print(f"Processor: \t{my_sys.processor}")
    print(f"Physical Core: \t{psutil.cpu_count(logical=False)}")
    print(f"Max CPU Freq: \t{cpufreq.max:.2f} Mhz")
    print(f"Memory: \t{str(round(psutil.virtual_memory().total / (1024.0 **
3)))} GB")
```

3.2.7 Plot.py

```
import matplotlib.pyplot as plt
```

```
def result_plot(points, closest_pair, dimension):
    # I.S. points is a list of tuples, closest_pair is a tuple of two tuples,
and dimension is an integer
    # F.S. Plot given points in blue and the closest pair colored in red and
draw a line between them in given dimension
    if dimension == 2:
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.scatter(*zip(*points))
        ax.scatter(*closest_pair[0], color='red')
        ax.scatter(*closest_pair[1], color='red')
        ax.plot([closest_pair[0][0], closest_pair[1][0]], [closest_pair[0][1],
closest_pair[1][1]], color='red')
        plt.show()

    elif dimension == 3:
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(*zip(*points))
        ax.scatter(*closest_pair[0], color='red')
        ax.scatter(*closest_pair[1], color='red')
        ax.plot([closest_pair[0][0], closest_pair[1][0]], [closest_pair[0][1],
closest_pair[1][1]], [closest_pair[0][2], closest_pair[1][2]], color='red')
        plt.show()

    else:
        print('Plotting is only available in 2D and 3D\n')
```

BAB IV

MASUKAN DAN LUARAN PROGRAM

4.1 Generate random

4.1.1 n = 16, d = 3

```
THE CLOSEST-PAIR PROBLEM SOLVER

              by

 Ditra Rizqa Amadia (13521019)
     Ahmad Nadil (13521025)

Masukkan banyaknya titik: 16
Masukkan dimensi: 3

== HARDWARE SPECIFICATION ==========----------
System:         Linux
Node name:      ditra-OMEN
Version:        #33~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Mon Jan 30 17:03:34 UTC 2
Machine:        x86_64
Processor:      x86_64
Physical Core:  8
Max CPU Freq:   4600.00 Mhz
Memory:         15 GB

== BRUTE FORCE ==========----------
Closest pair of points:
Point 1 :               [ 599 -896  337]
Point 2 :               [ 555 -908  130]
Distance:               211.9646196892302
Execution time:         0.0002 s
Euclidean Distance Operations: 128

== DIVIDE AND CONQUER ==========----------
Closest pair of points:
Point 1 :               [ 555 -908  130]
Point 2 :               [ 599 -896  337]
Distance:               211.9646196892302
Execution time:         0.0002 s
Euclidean Distance Operations: 27

== SOLUTIONS COMPARISON ==========----------
Solutions match
Brute force is faster by 0.0000 s
Divide and conquer used 101 less euclidean distance operations
```

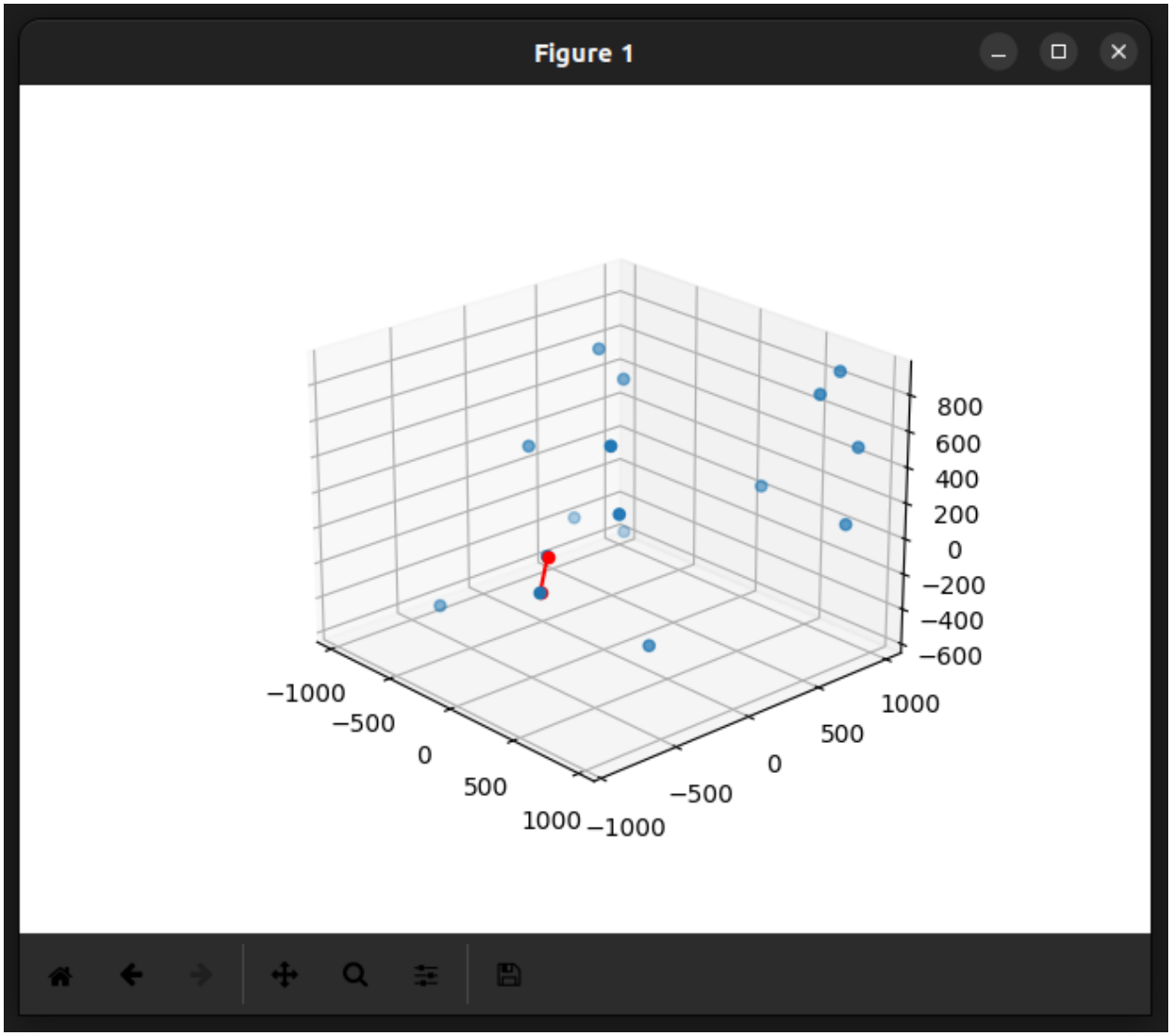


Figure 1
800
600
400
200
0
−200
−400
−600
−1000
−500
0
500
1000
−1000
−500
0
500
1000

4.1.2 n = 64, d = 3

```
THE CLOSEST-PAIR PROBLEM SOLVER

              by

 Ditra Rizqa Amadia (13521019)
     Ahmad Nadil (13521025)

Masukkan banyaknya titik: 64
Masukkan dimensi: 3

== HARDWARE SPECIFICATION ==========----------
System:         Linux
Node name:      ditra-OMEN
Version:        #33~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Mon Jan 30 17:03:34 UTC 2
Machine:        x86_64
Processor:      x86_64
Physical Core:  8
Max CPU Freq:   4600.00 Mhz
Memory:         15 GB

== BRUTE FORCE ==========----------
Closest pair of points:
Point 1 :               [  55 -779  347]
Point 2 :               [  76 -689  267]
Distance:               122.23338332877806
Execution time:         0.0103 s
Euclidean Distance Operations: 2019

== DIVIDE AND CONQUER ==========----------
Closest pair of points:
Point 1 :               [  55 -779  347]
Point 2 :               [  76 -689  267]
Distance:               122.23338332877806
Execution time:         0.0015 s
Euclidean Distance Operations: 235

== SOLUTIONS COMPARISON ==========----------
Solutions match
Divide and conquer is faster by 0.0088 s
Divide and conquer used 1784 less euclidean distance operations
```

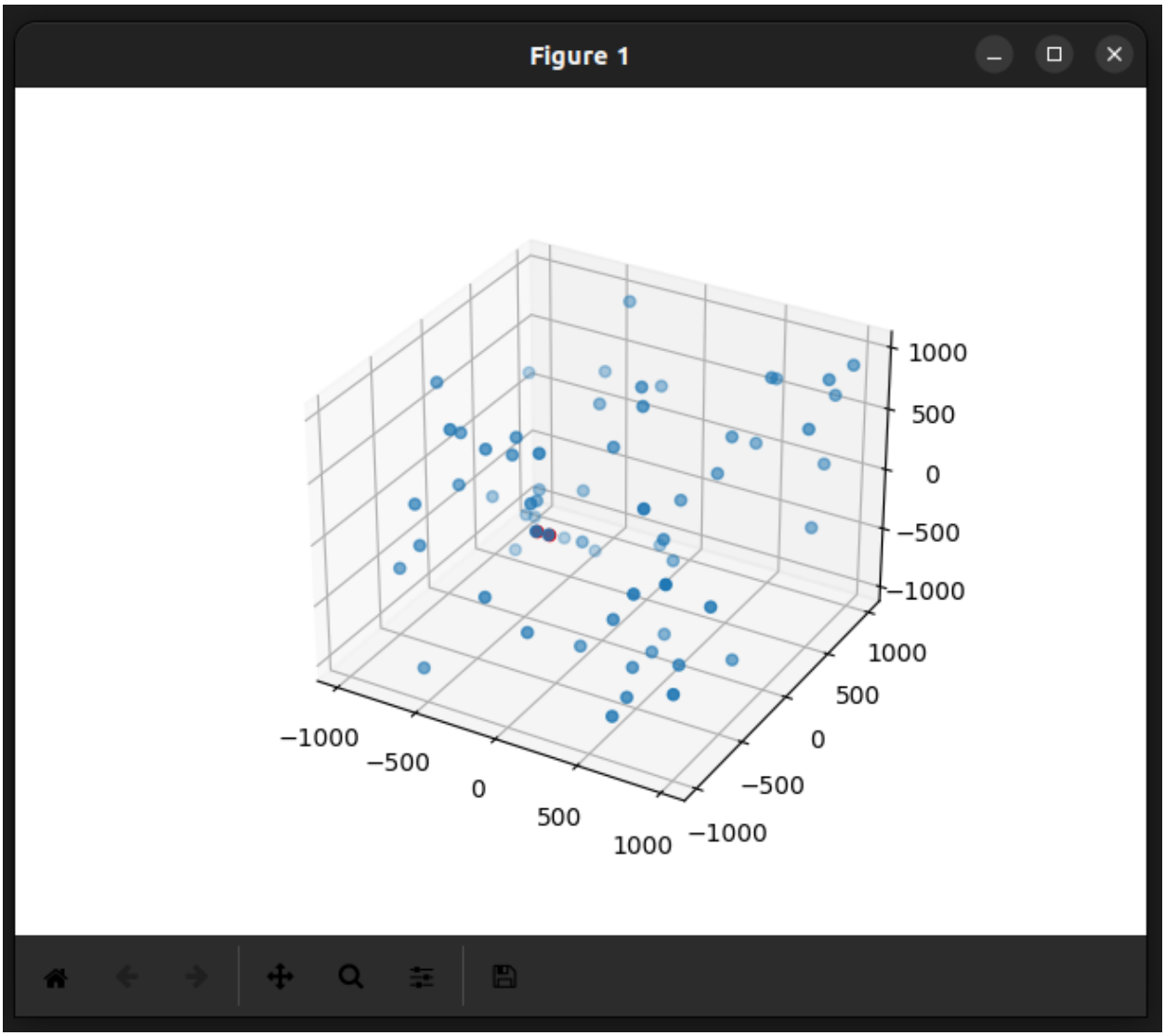


Figure 1
1000
500
0
−500
−1000
1000
500
0
−500
−1000
−1000
−500
0
500
1000

4.1.3 n = 128, d = 3

```
THE CLOSEST-PAIR PROBLEM SOLVER

              by

 Ditra Rizqa Amadia (13521019)
     Ahmad Nadil (13521025)

Masukkan banyaknya titik: 128
Masukkan dimensi: 3

== HARDWARE SPECIFICATION ==========----------
System:         Linux
Node name:      ditra-OMEN
Version:        #33~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Mon Jan 30 17:03:34 UTC 2
Machine:        x86_64
Processor:      x86_64
Physical Core:  8
Max CPU Freq:   4600.00 Mhz
Memory:         15 GB

== BRUTE FORCE ==========----------
Closest pair of points:
Point 1 :               [956  26 855]
Point 2 :               [938  10 825]
Distance:               38.47076812334269
Execution time:         0.0151 s
Euclidean Distance Operations: 8140

== DIVIDE AND CONQUER ==========----------
Closest pair of points:
Point 1 :               [938  10 825]
Point 2 :               [956  26 855]
Distance:               38.47076812334269
Execution time:         0.0028 s
Euclidean Distance Operations: 475

== SOLUTIONS COMPARISON ==========----------
Solutions match
Divide and conquer is faster by 0.0123 s
Divide and conquer used 7665 less euclidean distance operations
```

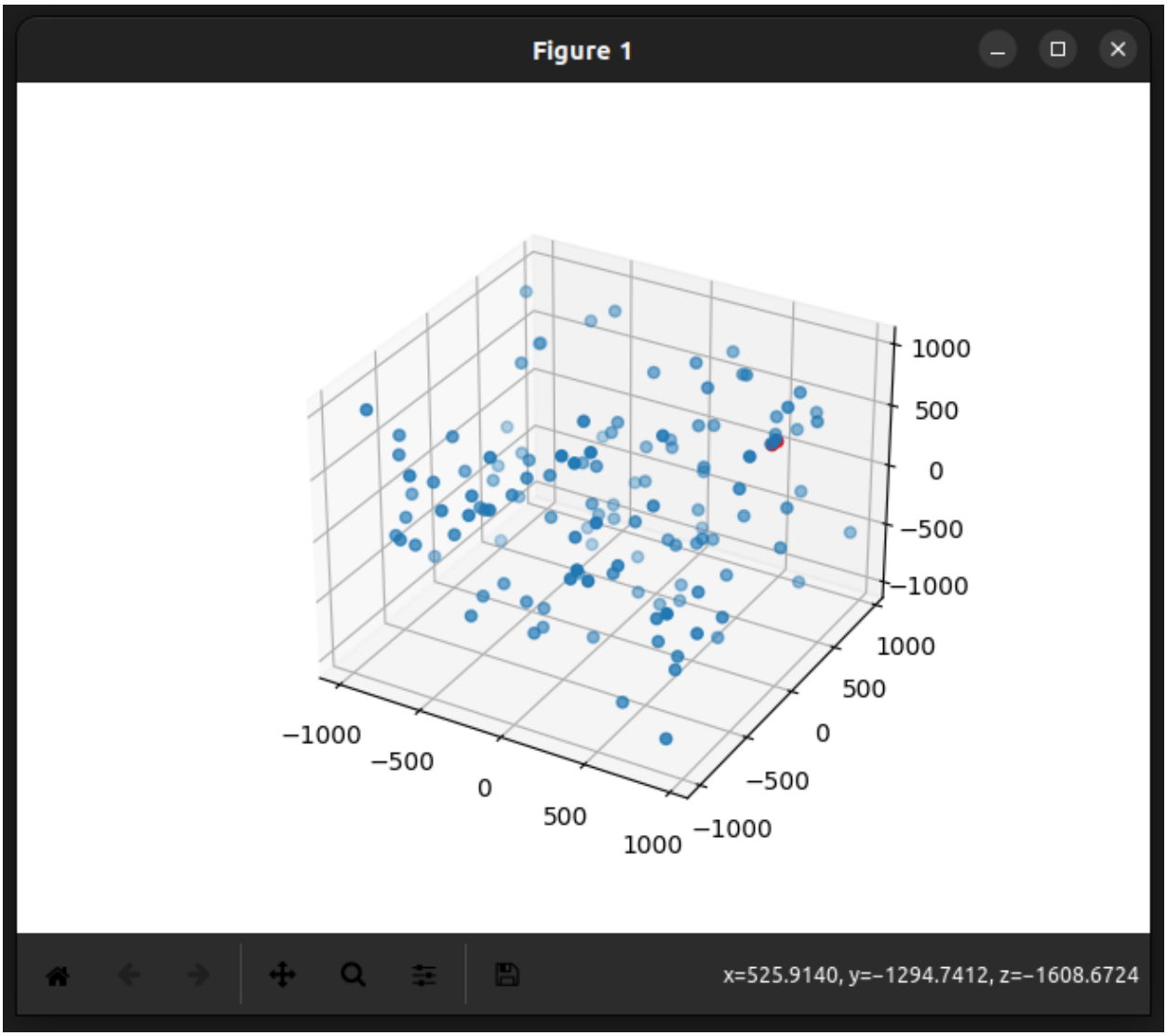


Figure 1
1000
500
0
−500
−1000
1000
500
0
−500
−1000
−1000
−500
0
500
1000
x=525.9140, y=−1294.7412, z=−1608.6724

4.1.4 n = 1000, d = 3

```
THE CLOSEST-PAIR PROBLEM SOLVER

              by

 Ditra Rizqa Amadia (13521019)
     Ahmad Nadil (13521025)

Masukkan banyaknya titik: 1000
Masukkan dimensi: 3

== HARDWARE SPECIFICATION ==========----------
System:         Linux
Node name:      ditra-OMEN
Version:        #33~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Mon Jan 30 17:03:34 UTC 2
Machine:        x86_64
Processor:      x86_64
Physical Core:  8
Max CPU Freq:   4600.00 Mhz
Memory:         15 GB

== BRUTE FORCE ==========----------
Closest pair of points:
Point 1 :               [ 688 -425  707]
Point 2 :               [ 690 -426  709]
Distance:               3.0
Execution time:         0.6417 s
Euclidean Distance Operations: 499513

== DIVIDE AND CONQUER ==========----------
Closest pair of points:
Point 1 :               [ 690 -426  709]
Point 2 :               [ 688 -425  707]
Distance:               3.0
Execution time:         0.0273 s
Euclidean Distance Operations: 5146

== SOLUTIONS COMPARISON ==========----------
Solutions match
Divide and conquer is faster by 0.6145 s
Divide and conquer used 494367 less euclidean distance operations
```

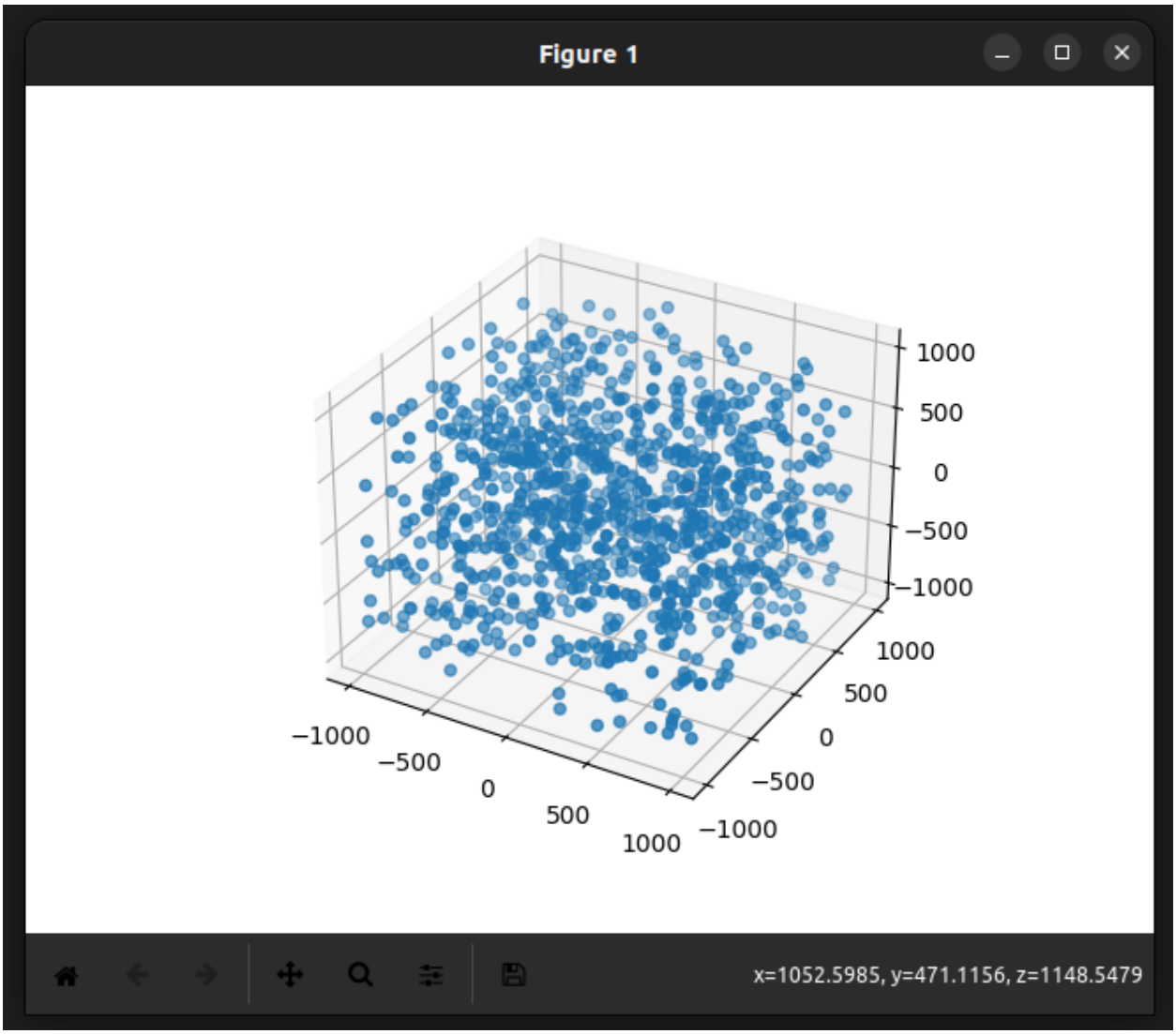


Figure 1
1000
500
0
−500
−1000
1000
500
0
−500
−1000
−1000
−500
0
500
1000
x=1052.5985, y=471.1156, z=1148.5479

4.1.5 n = 1000, d = 7

```
THE CLOSEST-PAIR PROBLEM SOLVER

              by

 Ditra Rizqa Amadia (13521019)
     Ahmad Nadil (13521025)

Masukkan banyaknya titik: 1000
Masukkan dimensi: 7

== HARDWARE SPECIFICATION ==========----------
System:         Linux
Node name:      ditra-OMEN
Version:        #33~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Mon Jan 30 17:03:34 UTC 2
Machine:        x86_64
Processor:      x86_64
Physical Core:  8
Max CPU Freq:   4600.00 Mhz
Memory:         15 GB

== BRUTE FORCE ==========----------
Closest pair of points:
Point 1 :               [  16 -982 -310 -400 -614 -792   60]
Point 2 :               [ 104 -954 -454 -528 -432 -726  -13]
Distance:               297.41721537261424
Execution time:         0.9759 s
Euclidean Distance Operations: 499510

== DIVIDE AND CONQUER ==========----------
Closest pair of points:
Point 1 :               [  16 -982 -310 -400 -614 -792   60]
Point 2 :               [ 104 -954 -454 -528 -432 -726  -13]
Distance:               297.41721537261424
Execution time:         0.4090 s
Euclidean Distance Operations: 140630

== SOLUTIONS COMPARISON ==========----------
Solutions match
Divide and conquer is faster by 0.5669 s
Divide and conquer used 358880 less euclidean distance operations


Plotting is only available in 2D and 3D
```

4.1.6 n = 1000, d = 2

```
THE CLOSEST-PAIR PROBLEM SOLVER

              by

 Ditra Rizqa Amadia (13521019)
     Ahmad Nadil (13521025)

Masukkan banyaknya titik: 1000
Masukkan dimensi: 2

== HARDWARE SPECIFICATION ==========----------
System:         Linux
Node name:      ditra-OMEN
Version:        #33~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Mon Jan 30 17:03:34 UTC 2
Machine:        x86_64
Processor:      x86_64
Physical Core:  8
Max CPU Freq:   4600.00 Mhz
Memory:         15 GB

== BRUTE FORCE ==========----------
Closest pair of points:
Point 1 :               [ 117 -789]
Point 2 :               [ 116 -789]
Distance:               1.0
Execution time:         0.5126 s
Euclidean Distance Operations: 499516

== DIVIDE AND CONQUER ==========----------
Closest pair of points:
Point 1 :               [ 116 -789]
Point 2 :               [ 117 -789]
Distance:               1.0
Execution time:         0.0175 s
Euclidean Distance Operations: 1771

== SOLUTIONS COMPARISON ==========----------
Solutions match
Divide and conquer is faster by 0.4951 s
Divide and conquer used 497745 less euclidean distance operations
```

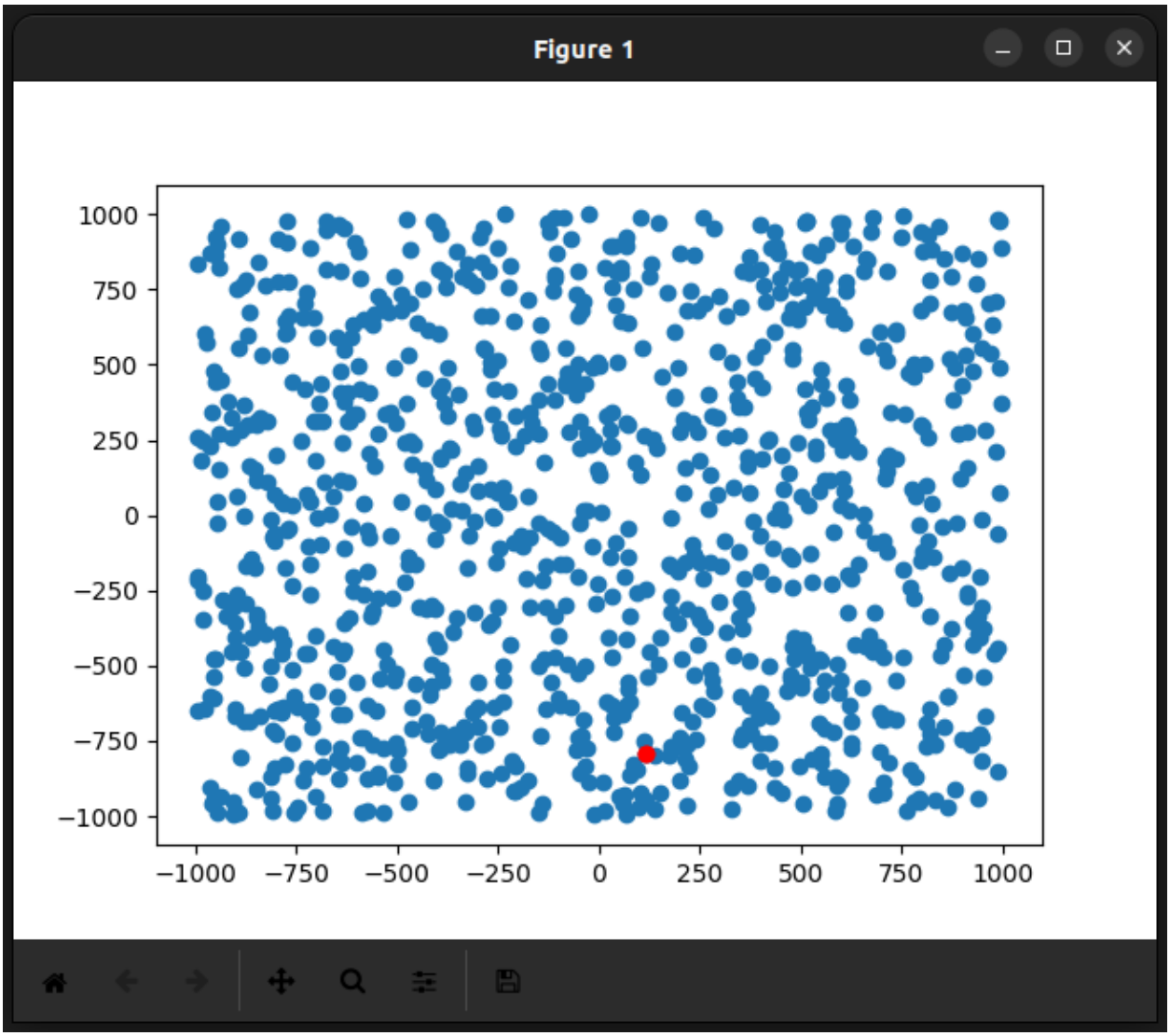


Figure 1
1000
750
500
250
0
−250
−500
−750
−1000
−1000
−750
−500
−250
0
250
500
750
1000

BAB V

LAMPIRAN

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

REFERENSI

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Sthttp://24solver.us-west-2.elasticbeanstalk.com/mik/2021-2022/Algoritma-Brute-Force-(2022)-Bag2.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf

Levitin, Anany, Introduction to The Design and Analysis of Algorithms, 3rd ed, USA: Addison-Wesley, 2012, pp. 169-197