LAPORAN TUGAS KECIL 2
MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA *DIVIDE AND CONQUER*

Disusun oleh:
1. 13521074 - Eugene Yap Jin Quan
2. 13521100 - Alexander Jason

Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S.T, M.Sc.
IF2211 - Strategi Algoritma

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

DAFTAR ISI

DAFTAR ISI **2**

BAB 1
DESKRIPSI MASALAH **2**

BAB 2
TEORI DASAR **3**

2.1. Algoritma Brute Force 3

2.2. Algoritma Divide and Conquer 3

2.3. Closest Pair 4

BAB 3
IMPLEMENTASI PROGRAM **6**

3.1. File: point_set.py 6

3.2. File: solver.py 8

3.3. File: visualizer.py 11

3.4. FIle: main.py 13

BAB 4
EKSPERIMEN **14**

4.1. N = 16 14

4.2. N = 64 15

4.3. N = 128 15

4.4. N = 1000 17

4.5. Keterangan 19

4.6. Analisis 21

BAB 5
PENUTUP **21**

5.1. Kesimpulan 21

5.2. Saran 24

5.3. Komentar dan Refleksi 24

5.4. Tabel Checkpoint 24

DAFTAR PUSTAKA **24**

BAB 1
DESKRIPSI MASALAH

Misalkan terdapat n buah titik pada ruang 3D. Setiap titik P di dalam ruang dinyatakan dengan koordinat P = (x, y, z). Solusi yang dicari adalah pasangan titik yang mempunyai jarak terdekat satu sama lain. Perhitungan jarak dua buah titik P1 = (x1, y1, z1) dan P2 = (x2, y2, z2) menggunakan rumus Euclidean berikut:

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

Pada Tugas Kecil ini, kami diminta untuk mengembangkan algoritma mencari pasangan titik dengan jarak terdekat pada bidang 3D. Algoritma ini dikembangkan dari *Algoritma Divide and Conquer* untuk mencari pasangan titik terdekat pada bidang 2D, yang telah diajarkan dalam kuliah IF2211 Strategi Algoritma.

Batasan dari implementasi adalah sebagai berikut:

- Implementasi menggunakan bahasa C/C++/Java/Python/Golang/Ruby/Perl.
- Solusi menggunakan algoritma *divide & conquer* dan dibandingkan dengan solusi yang menggunakan algoritma *brute force*.
- Masukan program adalah *n*, yaitu jumlah titik yang akan dibandingkan secara acak.
- Luaran program adalah solusi pasangan titik 3D dan jarak terdekat, jumlah penggunaan rumus jarak Euclidean, dan waktu pemrosesan.
- Implementasi boleh menyertakan visualisasi kumpulan titik 3D beserta solusi pasangan dengan jarak terdekat.
- Program juga boleh diimplementasikan untuk menyelesaikan masalah pasangan titik terdekat dalam ruang R^n

Untuk Tugas Kecil ini, implementasi kami menggunakan bahasa Python.

BAB 2
TEORI DASAR

2.1. Algoritma *Brute Force*

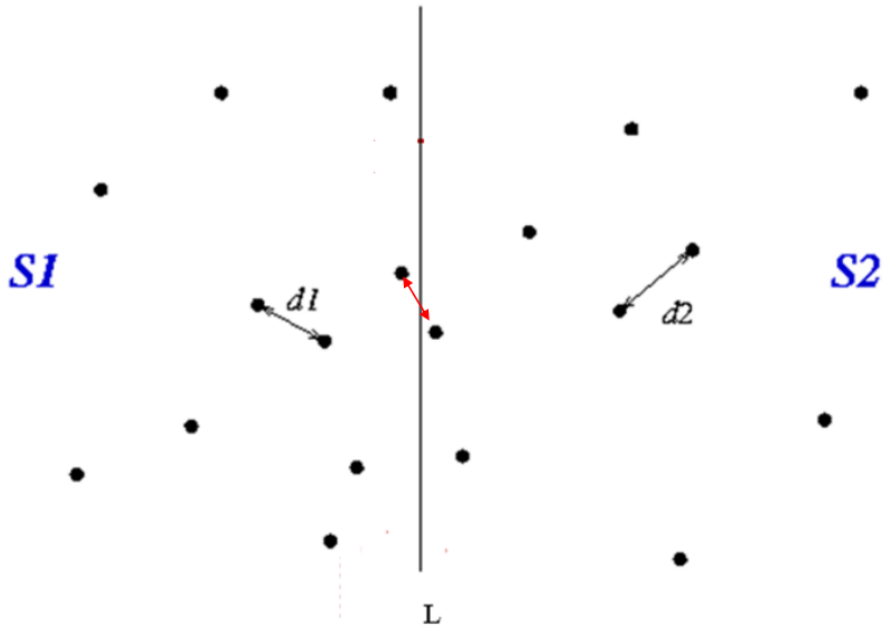
Salah satu algoritma yang digunakan dalam mencari *closest pair* dari sejumlah pasangan titik adalah algoritma *Brute Force.* Program akan mengecek jarak Euclidean dari semua pasangan titik yang ada. Kemudian program akan mencari jarak minimal dari setiap jarak yang sudah dihitung. Setelah menemukan jarak minimal, maka pasangan titik dengan jarak minimal tersebut adalah solusinya

2.2. Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* adalah algoritma pemecahan masalah yang menggunakan strategi membagi sebuah permasalahan besar menjadi bagian-bagian permasalahan yang lebih kecil secara rekursif. Permasalahan yang lebih kecil tersebut kemudian dicari solusinya kemudian digabungkan dengan solusi dari bagian permasalahan kecil lainnya sehingga menjadi sebuah solusi akhir.

2.3. Closest Pair

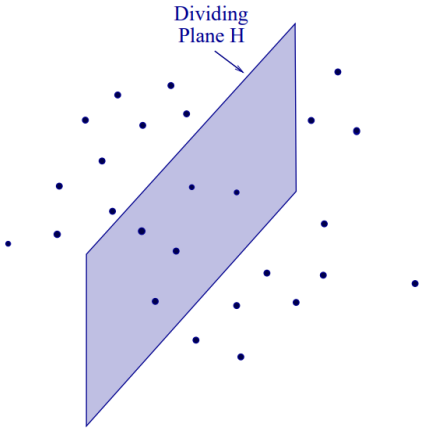
Closest Pair adalah sebuah persoalan dimana terdapat sebuah himpunan titik yang terdiri dari *n* buah titik pada bidang berdimensi *d* (2, 3, dst) untuk mencari sepasang titik dengan jarak terdekat. Secara *Brute Force,* program akan menghitung jarak dari setiap pasangan yang ada. Kemudian program akan memilih pasangan titik dengan jarak terkecil. Program tersebut memiliki kompleksitas algoritma $O(n^2)$.



Gambar 2.3.1 Permasalahan Closest Pair dalam 2D. Sumber: Algoritma Divide and Conquer (2021) - Bagian2

Dengan *Divide and Conquer,* kompleksitas algoritma dapat diperkecil hingga O(n log n). Berikut langkah-langkah pemecahan persoalan *closest pair* dengan *Divide and Conquer*:

1) Mengurutkan titik dari yang paling kecil berdasarkan letak koordinat dimulai dari sumbu pertama (sumbu x). Jika ada 2 titik atau lebih yang memiliki sumbu axis yang sama, maka program akan mengurutkan berdasarkan sumbu kedua (sumbu y), sumbu ketiga (sumbu z), dan seterusnya. Langkah ini dapat dilakukan sebelum mengeksekusi algoritma pencarian solusi.
2) Jika jumlah titik pada himpunan kurang dari 4, lakukan pencarian solusi dengan algoritma *brute force* dan gabungkan solusi dengan upa-himpunan lainnya. Jika tidak, lanjut ke langkah 3.
3) Membagi himpunan titik menjadi dua bagian berdasarkan nilai tengah sumbu *x* dari himpunan tersebut. Apabila himpunan sudah terurut, nilai tengah dapat menggunakan titik median. Hasil dari tahap ini adalah sebuah hiperbidang (*hyperplane*) dari dimensi *d* (misal dimensi adalah 3, hiperbidang adalah sebuah bidang pembatas).



Gambar 2.3.2 Ilustrasi Hiperbidang Pembatas Himpunan Titik 3D. Sumber: Closest Pair

4) Secara rekursif, terapkan algoritma pada langkah 2-4 hingga setiap upa-himpunan tersisa dengan kurang dari 4 titik. Hasil dari tahap ini adalah solusi dari upa-himpunan kiri dan upa-himpunan kanan berupa pasangan titik dan jarak minimum.
5) Cari pasangan titik terdekat beserta jaraknya dari setiap upa-himpunan lalu gabungkan upa-himpunan kiri dan kanan. Hasil tahap ini adalah sebuah pasangan titik dan jarak minimum δ.
6) Lakukan pengecekan jarak antara titik-titik yang memenuhi syarat jarak titik menuju hiperbidang lebih kecil atau sama dengan δ.
7) Perbarui hasil solusi apabila terdapat pasangan titik di sekitar hiperbidang yang memiliki jarak yang sama atau lebih kecil dari δ.
8) Gabungkan seluruh solusi.

Di luar algoritma pengurutan, kompleksitas algoritma *divide and conquer* ini adalah $T(n)=2T(n/2)+cn = O(n \log n)$.

BAB 3
IMPLEMENTASI PROGRAM

3.1. File: point_set.py

```
"""
:file: point_set.py

set of points functions used in the closest pair problem
"""

import random
```

```
def add_point(p_set, point):
    """add new point to p_set

    :param p_set: set of point(s)
    :param point: tuple of numbers
    """
    p_set.append(point)
```

```
def add_rand_point(p_set, min_max_val, f_precision, dimension=3):
    """add unique random point to p_set

    :param p_set: set of point(s)
    :param min_max_val: determines the minimum and maximum value
for randomization
    :param f_precision: determines digit count of fractional part
    :param dimension: dimension of point to be added, default = 3
    """
    point = []
    for i in range(dimension):  # create new point
        point.append(round(random.uniform(-min_max_val,
min_max_val), f_precision))

    if tuple(point) in p_set:  # ensure p_set element uniqueness
        add_rand_point(p_set, min_max_val, f_precision, dimension)
    else:
        add_point(p_set, tuple(point))
```

```
def add_n_rand_point(p_set, n, min_max_val, f_precision,
dimension=3):
```

```
    """add n amount of randomized points to p_set

    :param p_set: set of point(s)
    :param n: amount of points to be added
    :param min_max_val: determines the minimum and maximum value
for randomization
    :param f_precision: determines digit count of fractional part
    :param dimension: dimension of point to be added, default = 3
    """
    for i in range(n):
        add_rand_point(p_set, min_max_val, f_precision, dimension)
```

```
def print_point_set_info(p_set):
    """print information about point set

    :param p_set: set of point(s)
    """
    p_count = len(p_set)

    print("POINT SET INFORMATION:")
    print("Number of Points : " + str(p_count))
    if p_count > 0:
        print("Dimension : " + str(len(p_set[0])))
        print("Points :")
        for points in p_set:
            print(" " + str(points))
```

3.2. File: solver.py

```
"""
:file: solver.py

contains definitions of functions & procedures used in solving the
closest pair problem
"""

import math
```

```
def euclid_distance(point1, point2):
    """calculate euclidean distance between two points
```

```
    :param point1: tuple of numbers
    :param point2: tuple of numbers
    :return: euclidean distance of both points
    """
    return math.sqrt(sum((p1 - p2) ** 2 for p1, p2 in zip(point1,
point2)))
```

```
def merge_sort(arr):
    """sort an array using merge sort algorithm

    :param arr: array
    :return: sorted array
    """
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    l_arr = arr[:mid]
    r_arr = arr[mid:]

    l_arr = merge_sort(l_arr)
    r_arr = merge_sort(r_arr)

    return merge(l_arr, r_arr)
```

```
def merge(l_arr, r_arr):
    """merge process used in merge sort

    :param l_arr: left array of merge sort
    :param r_arr: right array of merge sort
    :return: sorted combined array
    """
    res = []
    i = 0
    j = 0

    while i < len(l_arr) and j < len(r_arr):
        if l_arr[i] < r_arr[j]:
            res.append(l_arr[i])
            i += 1
        else:
```

```
            res.append(r_arr[j])
            j += 1
    res += l_arr[i:]
    res += r_arr[j:]

    return res
```

```
def quick_sort(arr):
    """sort an array using quick sort algorithm

    :param arr: array
    :return: sorted array
    """
    if len(arr) <= 1:
        return arr
    pivot = arr[0]
    l_arr = []
    r_arr = []
    for i in range(1, len(arr)):
        if arr[i] < pivot:
            l_arr.append(arr[i])
        else:
            r_arr.append(arr[i])
    return quick_sort(l_arr) + [pivot] + quick_sort(r_arr)
```

```
def closest_pair_bf(points):
    """get the closest pair from point set using brute force

    :points: set (list) of points (tuple of numbers)
    :return: closest pair of points, minimum distance, euclidean
distance count
    """
    p_count = len(points)
    min_dist = float('inf')
    nearest_pair = []
    op_count = 0

    for i in range(p_count - 1):
        for j in range(i + 1, p_count):
            curr_dist = euclid_distance(points[i], points[j])
```

```
            op_count += 1
            if curr_dist < min_dist:
                min_dist = curr_dist
                nearest_pair = []
            if curr_dist <= min_dist:
                nearest_pair.append((points[i], points[j]))

    return nearest_pair, min_dist, op_count
```

```
def closest_pair_dnc(points):
    """get the closest pair from point set using divide and conquer

    :points: set (list) of points (tuple of numbers)
    :return: the closest pair of points, minimum distance,
euclidean distance count
    """
    p_count = len(points)
    dimension = len(points[0])
    if p_count <= 3:
        return closest_pair_bf(points)
    else:
        i_median = p_count // 2
        median_point = points[i_median]
        pair1, dist1, euclid_count1 =
closest_pair_dnc(points[:i_median])
        pair2, dist2, euclid_count2 =
closest_pair_dnc(points[i_median:])

        min_dist = min(dist1, dist2)
        if dist1 != dist2:
            min_pair = pair1 if min_dist == dist1 else pair2
        else:
            min_pair = pair1 + pair2

        total_euclid_count = euclid_count1 + euclid_count2

        delta_strip = []
        for p in points:
            if abs(p[0] - median_point[0]) < min_dist:
                delta_strip.append(p)

        delta_count = len(delta_strip)
```

```
        for i in range(delta_count):
            for j in range(i + 1, delta_count):
                within_delta = True
                for axis in range(1, dimension):
                    within_delta = within_delta and
abs(delta_strip[i][axis] - delta_strip[j][axis]) < min_dist

                if within_delta:
                    dist3 = euclid_distance(delta_strip[i],
delta_strip[j])
                    total_euclid_count += 1

                    if dist3 < min_dist:
                        min_dist = dist3
                        min_pair = []
                    if dist3 <= min_dist:
                        min_pair.append((delta_strip[i],
delta_strip[j]))

        return min_pair, min_dist, total_euclid_count
```

```
def display_solution(solution_set, distance, euclid_count,
solve_time):
    """prints solution

    :param solution_set: set of solution pairs
    :param distance: minimum distance of the closes pair problem
    :param euclid_count: euclidean distance formula usage count
    :param solve_time: recorded processing time in ns
    """
    solution_set = set(solution_set)        # ensure unique
solutions on print
    print("Solution Pair(s) :")
    for solution in solution_set:
        print(" ", solution[0], "and", solution[1])
    print("Distance :", distance)
    print("Euclidean Distance Usage Count :", euclid_count)
    print("Processing time :", solve_time / 1000000, "ms")
```

3.3. File: visualizer.py

```
"""
:file: visualizer.py

visualize points and solution from the closest pair problem (2D/3D)
"""

from matplotlib import pyplot as plt
from point_set import *
```

```
def visualizer(p_dimension, p_set, pair_dnc):
    """visualize points and solution

    :param p_dimension: dimension of points
    :param p_set: set of points
    :param pair_dnc: solution of the closest pair problem (divide
and conquer)
    """
    fig = plt.figure(figsize=(100, 100))

    if p_dimension == 3:
        ax = fig.add_subplot(111, projection="3d")
        for i in range(len(p_set)):
            ax.scatter(p_set[i][0], p_set[i][1], p_set[i][2],
c='b', alpha=0.5, marker='o')
        for i in range(0, len(pair_dnc)):
            for j in range(0, len(pair_dnc[i])):
                ax.scatter(pair_dnc[i][j][0], pair_dnc[i][j][1],
pair_dnc[i][j][2], c='r', marker='o')

        for i in range(0, len(pair_dnc)):
            x1 = [pair_dnc[i][0][0], pair_dnc[i][1][0]]
            x2 = [pair_dnc[i][0][1], pair_dnc[i][1][1]]
            x3 = [pair_dnc[i][0][2], pair_dnc[i][1][2]]
            plt.plot(x1, x2, x3, 'ro-')

    if p_dimension == 2:
        for i in range(len(p_set)):
            plt.scatter(p_set[i][0], p_set[i][1], alpha=0.5,
color='b')
        for i in range(len(pair_dnc)):
            for j in range(0, len(pair_dnc[i])):
                plt.scatter(pair_dnc[i][j][0], pair_dnc[i][j][1],
color='r')
```

```
        # draw line
        for i in range(0, len(pair_dnc)):
            x1 = [pair_dnc[i][0][0], pair_dnc[i][1][0]]
            x2 = [pair_dnc[i][0][1], pair_dnc[i][1][1]]
            plt.plot(x1, x2, 'ro-')

        x = []
        y = []
        for i in range(len(p_set)):
            x.append(p_set[i][0])
            y.append(p_set[i][1])

    plt.show()
```

3.4. *FIle: main.py*

```
"""
:file: main.py

main program for closest pair problem
"""

from point_set import *
from solver import *
import time
from visualizer import *
```

```
def int_input_validation(val_min, val_max, prompt):
    """validate integer input

    :param val_min: minimum value for integer input (inclusive)
    :param val_max: maximum value for integer input (inclusive)
    :param prompt: input prompt
    :return: validated integer value
    """
    valid = False
    result = input(prompt)
    while not valid:
        try:
            result = int(result)
            if val_min <= result <= val_max:
                valid = True
```

```
            else:
                print("Input must be in between " + str(val_min) +
" and " + str(val_max) + "!")
                result = input(prompt)
        except:
            print("Input Invalid!")
            result = input(prompt)
    return result
```

```
# **** MAIN FUNCTION ****
p_set = []

# Input Validation
p_dimension = int_input_validation(2, 100, "Input Dimension : ")
p_count = int_input_validation(1, 10000, "Input Amount of Points :
")
min_max = int_input_validation(1, 10000, "Input Maximum & Minimum
Value for All Axes : ")
f_precision = int_input_validation(0, 10, "Input Fractional
Precision : ")

# Add Points and Sort by Each Axes
add_n_rand_point(p_set, p_count, min_max, f_precision, p_dimension)
p_set = quick_sort(p_set)

print()

# Display Point Set Information (Points, Dimension, Point Count)
show_info_choice = int_input_validation(0, 1, "Show Point Set
Information? (1/0) : ")
if show_info_choice == 1:
    print()
    print_point_set_info(p_set)

print()

# Solve by Brute Force and by Divide & Conquer
print("=======BRUTE FORCE SOLUTION========")
timer_bf = time.time_ns()
pair_bf, dist_bf, op_count_bf = closest_pair_bf(p_set)
timer_bf = time.time_ns() - timer_bf
display_solution(pair_bf, dist_bf, op_count_bf, timer_bf)
```

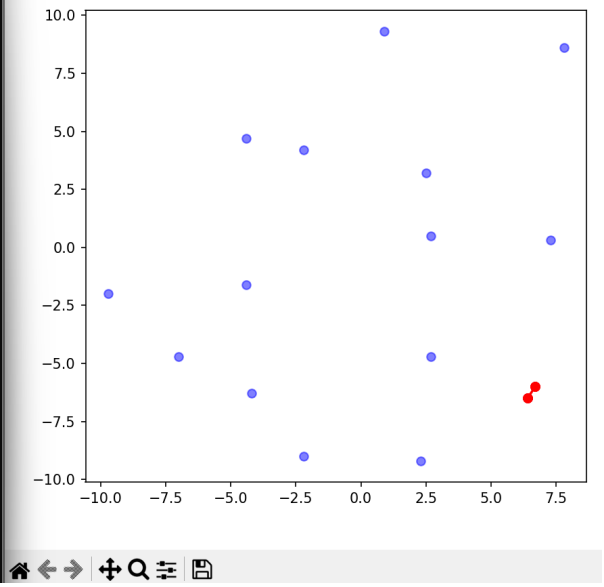
```
print()

print("=====DIVIDE & CONQUER SOLUTION=====")
timer_dnc = time.time_ns()
pair_dnc, dist_dnc, op_count_dnc = closest_pair_dnc(p_set)
timer_dnc = time.time_ns() - timer_dnc
display_solution(pair_dnc, dist_dnc, op_count_dnc, timer_dnc)

print()

# Validity Check
if dist_bf == dist_dnc:
    print("SOLUTION VALID")

    # Visualize only for 2 and 3 dimension
    if 1 < p_dimension < 4:
        print("VISUALIZING POINTS (2D/3D)")
        visualizer(p_dimension, p_set, pair_dnc)

else:
    print("SOLUTION INVALID")
```

BAB 4
EKSPERIMEN

4.1. N = 16

a. Dimensi 2

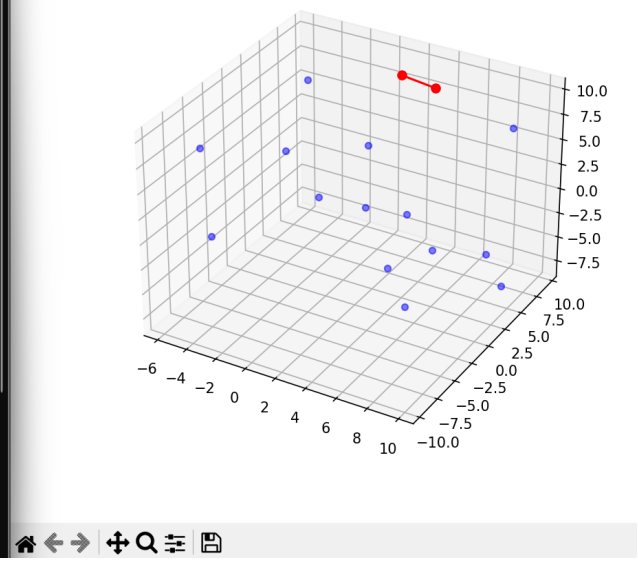
```
Input Amount of Points : 16
Input Maximum & Minimum Value for All Axes : 10
Input Fractional Precision : 1

Show Point Set Information? (1/0) : 1

POINT SET INFORMATION:
Number of Points : 16
Dimension : 2
Points :
 (-9.7, -2.0)
 (-7.0, -4.7)
 (-4.4, -1.6)
 (-4.4, 4.7)
 (-4.2, -6.3)
 (-2.2, -9.0)
 (-2.2, 4.2)
 (0.9, 9.3)
 (2.3, -9.2)
 (2.5, 3.2)
 (2.7, -4.7)
 (2.7, 0.5)
 (6.4, -6.5)
 (6.7, -6.0)
 (7.3, 0.3)
 (7.8, 8.6)

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (6.4, -6.5) and (6.7, -6.0)
Distance : 0.583095189484453
Euclidean Distance Usage Count : 120
Processing time : 0.0 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (6.4, -6.5) and (6.7, -6.0)
Distance : 0.583095189484453
Euclidean Distance Usage Count : 16
Processing time : 0.0 ms
```



Gambar 4.1.1 Percobaan n=16 dimensi 2

b. Dimensi 3

```
Input Amount of Points : 16
Input Maximum & Minimum Value for All Axes : 10
Input Fractional Precision : 1

Show Point Set Information? (1/0) : 1

POINT SET INFORMATION:
Number of Points : 16
Dimension : 3
Points :
 (-5.8, -4.0, -3.0)
 (-5.4, -5.9, 7.2)
 (-4.2, 6.2, 7.5)
 (-4.1, 2.8, 2.3)
 (1.2, 3.6, 4.6)
 (1.8, 7.1, 9.8)
 (3.3, 2.0, -6.1)
 (4.2, 6.7, -6.9)
 (4.2, 6.7, 9.7)
 (4.3, -9.5, 8.8)
 (4.5, 2.1, -0.1)
 (5.5, -5.9, 6.1)
 (6.7, 9.3, -7.9)
 (8.4, -6.7, -1.8)
 (9.7, 4.6, -6.7)
 (9.8, 5.5, 8.7)

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (1.8, 7.1, 9.8) and (4.2, 6.7, 9.7)
Distance : 2.43515913237718 44
Euclidean Distance Usage Count : 120
Processing time : 1.2504 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (1.8, 7.1, 9.8) and (4.2, 6.7, 9.7)
Distance : 2.43515913237718 44
Euclidean Distance Usage Count : 17
Processing time : 0.0 ms
```



Gambar 4.1.2 Percobaan n=16 dimensi 3

c. Dimensi 4

```
Input Dimension : 4
Input Amount of Points : 16
Input Maximum & Minimum Value for All Axes : 10
Input Fractional Precision : 1

Show Point Set Information? (1/0) : 1

POINT SET INFORMATION:
Number of Points : 16
Dimension : 4
Points :
 (-6.9, 6.0, -6.7, -8.3)
 (-6.6, -3.4, -0.4, 7.9)
 (-6.4, 4.4, 3.1, 6.4)
 (-4.6, -0.6, -5.4, 5.4)
 (-4.6, 1.1, 2.2, -6.3)
 (-3.3, 1.6, 0.6, -8.4)
 (-3.2, -1.2, 7.1, -4.2)
 (-1.8, -1.8, 1.1, -7.4)
 (-1.5, -6.2, -7.7, 4.4)
 (-1.1, 4.8, 8.6, -9.8)
 (1.3, 1.6, 9.6, -6.9)
 (4.3, 7.7, 8.6, 4.3)
 (5.2, -0.7, 4.8, -5.6)
 (5.4, 2.7, -5.7, 1.6)
 (5.9, -6.6, 1.6, 8.9)
 (8.5, -4.3, -1.2, 3.8)

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (-4.6, 1.1, 2.2, -6.3) and (-3.3, 1.6, 0.6, -8.4)
Distance : 2.98496231131986
Euclidean Distance Usage Count : 120
Processing time : 1.0215 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (-4.6, 1.1, 2.2, -6.3) and (-3.3, 1.6, 0.6, -8.4)
Distance : 2.98496231131986
Euclidean Distance Usage Count : 18
Processing time : 0.0 ms

SOLUTION VALID
```

Gambar 4.1.3 Percobaan n=16 dimensi 4

d. Dimensi 5

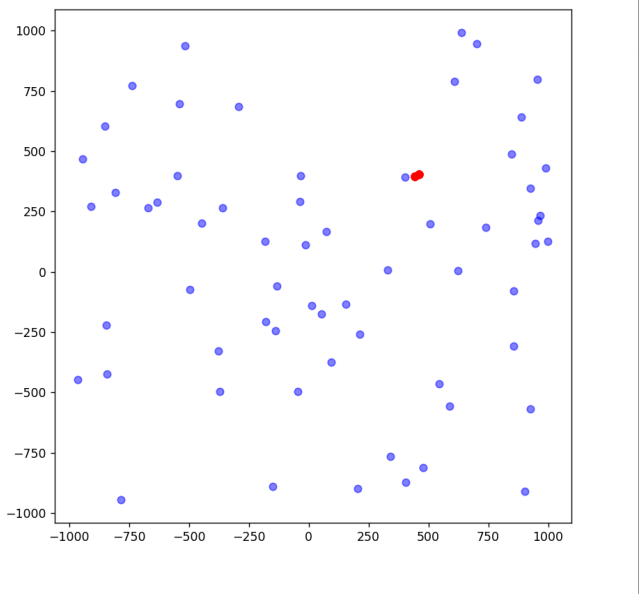
```
Input Amount of Points : 16
Input Maximum & Minimum Value for All Axes : 10
Input Fractional Precision : 1

Show Point Set Information? (1/0) : 1

POINT SET INFORMATION:
Number of Points : 16
Dimension : 5
Points :
 (-9.0, 4.4, -6.7, -9.5, -2.2)
 (-4.9, -5.4, 1.3, -7.2, -1.9)
 (-4.3, -8.8, 8.9, 3.5, 3.3)
 (-3.5, 4.6, 9.2, 4.9, 7.5)
 (-2.2, -6.4, -0.8, -1.8, -8.6)
 (-1.4, -6.5, 5.9, 3.8, -5.7)
 (1.6, -1.0, -8.5, 7.3, -8.9)
 (2.2, -1.7, -4.3, -9.0, 1.0)
 (3.5, -2.7, -8.6, 1.6, -4.2)
 (4.5, -0.8, -0.1, -3.5, -2.9)
 (4.7, 5.0, -2.3, 5.1, -7.6)
 (5.3, 0.4, 6.6, -5.9, -4.9)
 (5.5, 6.4, -8.3, 5.9, 5.0)
 (6.1, -2.3, 4.7, -8.6, 8.2)
 (7.4, 3.7, -5.1, 7.6, 9.2)
 (7.5, -0.2, -2.7, 0.7, -4.9)

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (4.5, -0.8, -0.1, -3.5, -2.9) and (7.5, -0.2, -2.7, 0.7, -4.9)
Distance : 6.144916598294887
Euclidean Distance Usage Count : 120
Processing time : 0.0 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (4.5, -0.8, -0.1, -3.5, -2.9) and (7.5, -0.2, -2.7, 0.7, -4.9)
Distance : 6.144916598294887
Euclidean Distance Usage Count : 37
Processing time : 0.0 ms

SOLUTION VALID
```

Gambar 4.1.4 Percobaan n=16 dimensi 5

4.2. N = 64

a. Dimensi 2

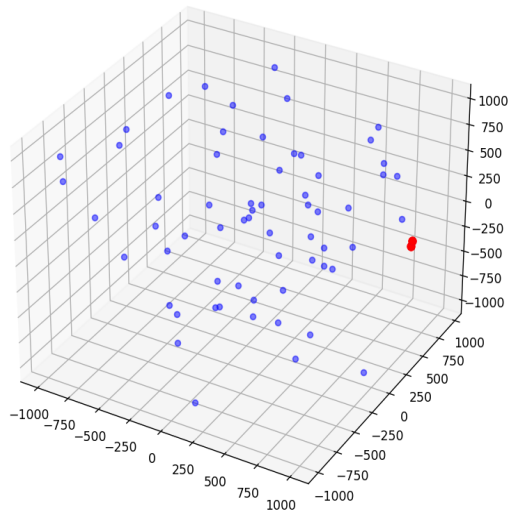
```
.\main.exe
Input Dimension : 2
Input Amount of Points : 64
Input Maximum & Minimum Value for All Axes : 1000
Input Fractional Precision : 2

Show Point Set Information? (1/0) : 0

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (443.06, 397.07) and (459.46, 404.44)
Distance : 17.97990266936947
Euclidean Distance Usage Count : 2016
Processing time : 2.9374 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (443.06, 397.07) and (459.46, 404.44)
Distance : 17.97990266936947
Euclidean Distance Usage Count : 60
Processing time : 0.9232 ms

SOLUTION VALID
VISUALIZING POINTS (2D/3D)
```



Gambar 4.2.1 Percobaan n=64 dimensi 2

b. Dimensi 3

```
\main.exe
Input Dimension : 3
Input Amount of Points : 64
Input Maximum & Minimum Value for All Axes : 1000
Input Fractional Precision : 2

Show Point Set Information? (1/0) : 0

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (957.27, 622.64, -193.76) and (994.94, 568.15, -92.24)
Distance : 121.22087031530505
Euclidean Distance Usage Count : 2016
Processing time : 5.0359 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (957.27, 622.64, -193.76) and (994.94, 568.15, -92.24)
Distance : 121.22087031530505
Euclidean Distance Usage Count : 74
Processing time : 0.9098 ms

SOLUTION VALID
VISUALIZING POINTS (2D/3D)
```



Gambar 4.2.2 Percobaan n=64 dimensi 3

c. Dimensi 4

```
Input Dimension : 4
Input Amount of Points : 64
Input Maximum & Minimum Value for All Axes : 1000
Input Fractional Precision : 2

Show Point Set Information? (1/0) : 0

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (132.95, -94.81, -109.44, -504.64) and (228.77, -100.27, -38.07, -503.1)
Distance : 119.61326222455436
Euclidean Distance Usage Count : 2016
Processing time : 4.9999 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (132.95, -94.81, -109.44, -504.64) and (228.77, -100.27, -38.07, -503.1)
Distance : 119.61326222455436
Euclidean Distance Usage Count : 102
Processing time : 1.0798 ms

SOLUTION VALID
```

Gambar 4.2.3 Percobaan n=64 dimensi 4

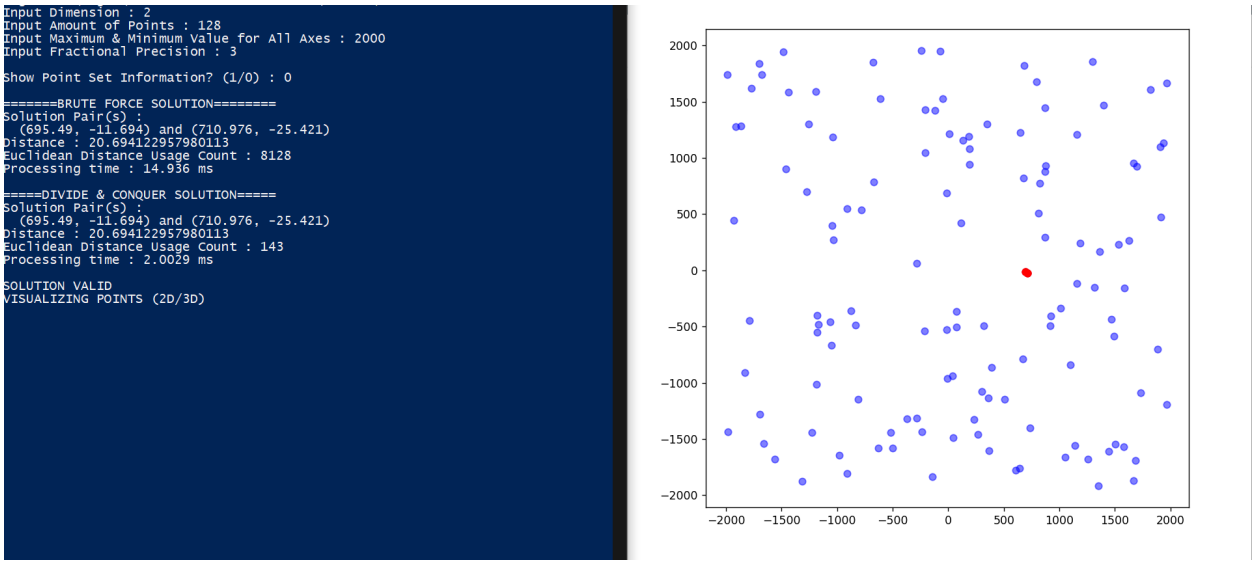
d. Dimensi 5

```
Input Dimension : 5
Input Amount of Points : 64
Input Maximum & Minimum Value for All Axes : 1000
Input Fractional Precision : 2

Show Point Set Information? (1/0) : 0

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (500.85, 632.48, -749.67, -67.03, 399.59) and (550.16, 640.71, -678.36, -79.57, 290.06)
Distance : 140.49340767452395
Euclidean Distance Usage Count : 2016
Processing time : 6.0356 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (500.85, 632.48, -749.67, -67.03, 399.59) and (550.16, 640.71, -678.36, -79.57, 290.06)
Distance : 140.49340767452395
Euclidean Distance Usage Count : 146
Processing time : 4.9209 ms

SOLUTION VALID
```

Gambar 4.2.4 Percobaan n=64 dimensi 5

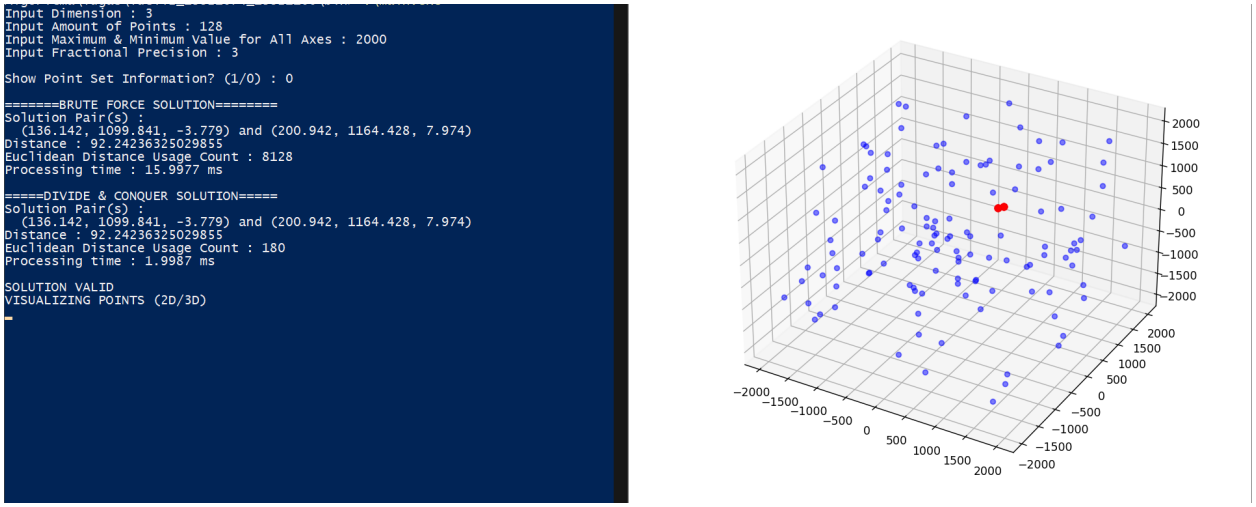
4.3. N = 128

a. Dimensi 2



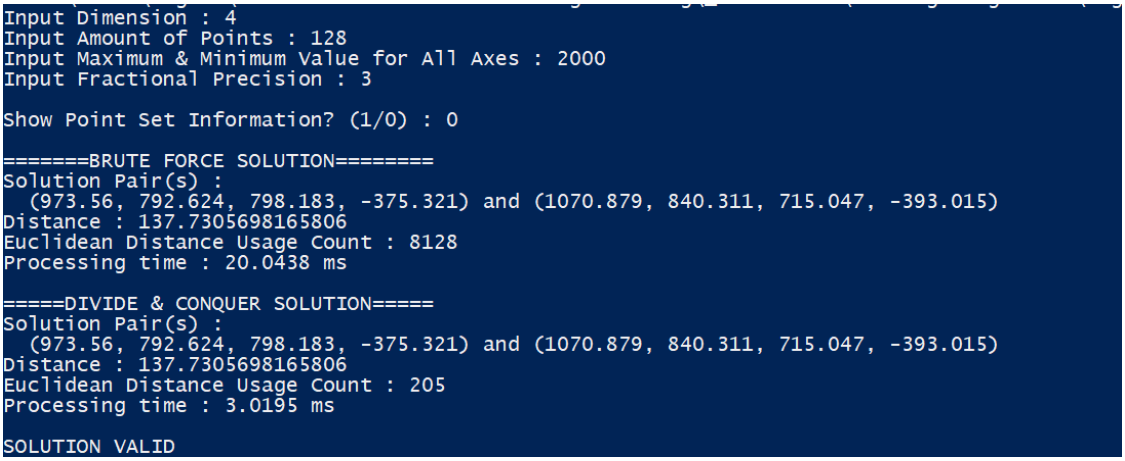
Gambar 4.3.1 Percobaan n=128 dimensi 2

b. Dimensi 3



Gambar 4.3.2 Percobaan n=128 dimensi 3

c. Dimensi 4



Gambar 4.3.3 Percobaan n=128 dimensi 4

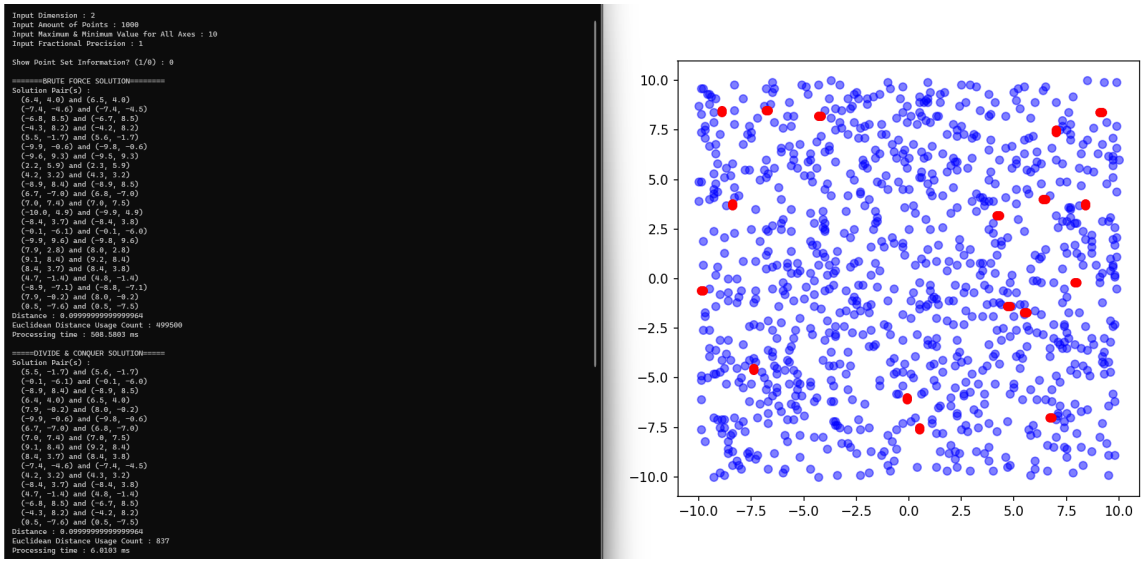
d. Dimensi 5

```
Input Dimension : 5
Input Amount of Points : 128
Input Maximum & Minimum Value for All Axes : 2000
Input Fractional Precision : 3

Show Point Set Information? (1/0) : 0

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (-1036.705, 1941.036, 1068.724, -899.895, -1941.135) and (-1020.242, 1877.073, 1116.412, -760.573, -1664.67)
Distance : 320.1249084201352
Euclidean Distance Usage Count : 8128
Processing time : 27.9983 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (-1036.705, 1941.036, 1068.724, -899.895, -1941.135) and (-1020.242, 1877.073, 1116.412, -760.573, -1664.67)
Distance : 320.1249084201352
Euclidean Distance Usage Count : 265
Processing time : 4.032 ms

SOLUTION VALID
```

Gambar 4.3.4 Percobaan n=128 dimensi 4

4.4. N = 1000

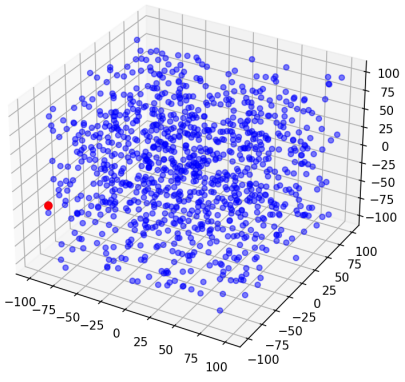
a. Dimensi 2



Gambar 4.4.1 Percobaan n=1000 dimensi 2

b. Dimensi 3

```
Input Dimension : 3
Input Amount of Points : 1000
Input Maximum & Minimum Value for All Axes : 100
Input Fractional Precision : 1

Show Point Set Information? (1/0) : 0

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (-99.1, -81.7, -34.4) and (-98.7, -82.7, -34.3)
Distance : 1.0816653826391938
Euclidean Distance Usage Count : 499500
Processing time : 578.5917 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (-99.1, -81.7, -34.4) and (-98.7, -82.7, -34.3)
Distance : 1.0816653826391938
Euclidean Distance Usage Count : 1406
Processing time : 21.0492 ms

SOLUTION VALID
VISUALIZING POINTS (2D/3D)
```



Gambar 4.4.2 Percobaan n=1000 dimensi 3

c. Dimensi 4

```
Input Dimension : 4
Input Amount of Points : 1000
Input Maximum & Minimum Value for All Axes : 100
Input Fractional Precision : 1

Show Point Set Information? (1/0) : 0

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (38.2, 32.5, 24.4, -98.3) and (38.7, 29.9, 20.8, -95.9)
Distance : 5.072474741188954
Euclidean Distance Usage Count : 499500
Processing time : 670.2901 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (38.2, 32.5, 24.4, -98.3) and (38.7, 29.9, 20.8, -95.9)
Distance : 5.072474741188954
Euclidean Distance Usage Count : 1657
Processing time : 34.3482 ms

SOLUTION VALID
```

Gambar 4.4.3 Percobaan n=1000 dimensi 4

d. Dimensi 5

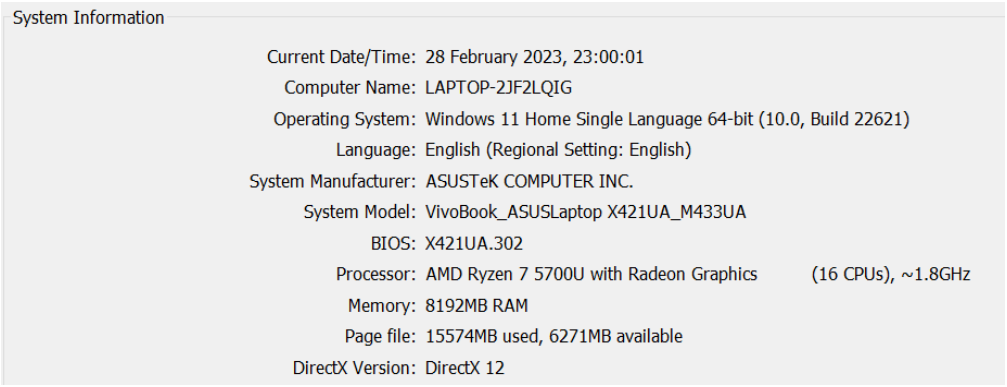
```
Input Dimension : 5
Input Amount of Points : 1000
Input Maximum & Minimum Value for All Axes : 100
Input Fractional Precision : 1

Show Point Set Information? (1/0) : 0

=======BRUTE FORCE SOLUTION========
Solution Pair(s) :
  (74.7, -24.0, -85.7, 18.2, -89.4) and (74.9, -23.2, -81.2, 11.1, -95.4)
Distance : 10.360501918343532
Euclidean Distance Usage Count : 499500
Processing time : 761.5142 ms

=====DIVIDE & CONQUER SOLUTION=====
Solution Pair(s) :
  (74.7, -24.0, -85.7, 18.2, -89.4) and (74.9, -23.2, -81.2, 11.1, -95.4)
Distance : 10.360501918343532
Euclidean Distance Usage Count : 2027
Processing time : 60.7803 ms

SOLUTION VALID
```

Gambar 4.4.4 Percobaan n=1000 dimensi 5

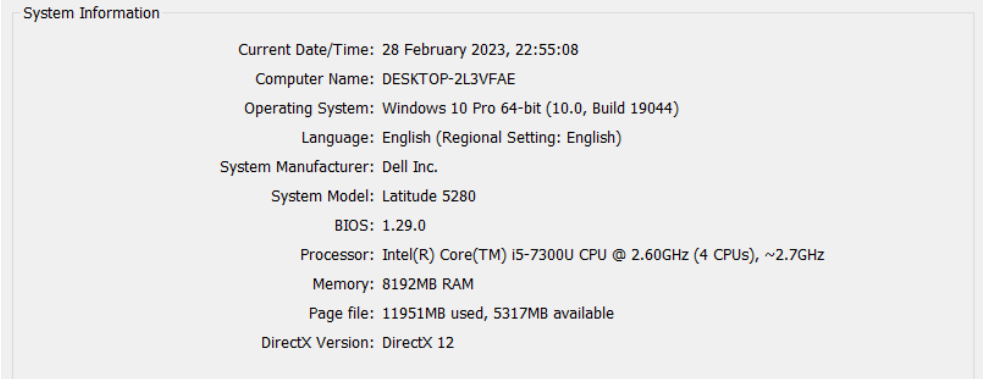
4.5. Keterangan Tambahan

- N merupakan jumlah titik pada himpunan titik P.

- Visualisasi hanya dilakukan untuk dimensi 2 dan dimensi 3.
- Untuk percobaan N = 16 dan N = 1000, program dijalankan pada laptop *Asus Vivobook S16* dengan spesifikasi berikut.



- Untuk percobaan N = 64 dan N = 128, program dijalankan pada laptop *Dell Latitude 5280* dengan spesifikasi berikut.



4.6. Analisis

Pada n=16, *execution time* pada kedua algoritma sama. Namun ketika jumlah n semakin banyak, perbedaan *execution time* kedua algoritma semakin tinggi. Hal ini disebabkan karena pada algoritma *Brute Force* program akan menghitung jarak dari semua pasangan titik. Namun, pada algoritma *Divide and Conquer* program membagi himpunan titik menjadi upa-persoalan yang lebih kecil. Konsekuensi dari ukuran persoalan yang kecil adalah jumlah perhitungan yang lebih sedikit, baik pada pemanggilan *Brute Force* maupun pada pengecekan partisi δ. Hal ini terbukti pada jumlah perhitungan jarak Euclidean pada setiap eksperimen.

BAB 5
PENUTUP

5.1. Kesimpulan

Dalam mencari *closest pair* dari sejumlah pasangan titik yang ada, dapat menggunakan algoritma *Brute Force* maupun *Divide and Conquer.* Dari hasil analisis dan implementasi menggunakan kedua algoritma tersebut, didapat bahwa pencarian solusi dengan algoritma *Divide and Conquer* memiliki performa yang lebih cepat dibandingkan solusi dengan algoritma *Brute Force* secara murni. Perbedaan performa dari kedua solusi tampak pada pengujian kami, dan perbedaan ini bersifat ekstrim ketika jumlah titik banyak.

5.2. Saran

Implementasi pencarian *closest pair* yang kami buat masih dapat dikembangkan lebih lanjut. Dari segi performa, implementasi algoritma *divide and conquer* kami juga masih dapat ditingkatkan, seperti dengan menggantikan bahasa pemrograman yang digunakan.

5.3. Komentar dan Refleksi

Kami senang karena pelajaran yang diberikan dosen di kelas (Bu Ulfa) dapat kami implementasikan dengan baik pada tugas kali ini. Kami berterima kasih kepada Bu Ulfa sebagai dosen pengampu kelas K2 serta Pak Rinaldi yang memberikan materi secara lengkap dan jelas.

5.4. Tabel *Checkpoint*

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	✓	

DAFTAR PUSTAKA

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tucil2-Stima-2023.pdf
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf
https://sites.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf

LAMPIRAN

LINK REPOSITORY

Link repository GitHub : https://github.com/yuujin-Q/Tucil2_13521074_13521100

PEMBAGIAN TUGAS

NIM	Nama	Tugas
13521074	Eugene Yap Jin Quan	Setup, solver, laporan
13521100	Alexander Jason	Sort, visualizer, laporan