LAPORAN TUGAS KECIL II

IF2211 STRATEGI ALGORITMA

PENCARIAN PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA DIVIDE AND CONQUER

Disusun oleh :

Rachel Gabriela Chen (13521044)

Bill Clinton (13521064)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022/2023

DAFTAR ISI

1. Pendahuluan

Setiap titik P dalam ruang 3D dapat dinyatakan dengan koordinat P = (x, y, z). Jarak dari setiap dua titik (Misalkan titik A = (x_1, y_1, z_1) dan titik B = (x_2, y_2, z_2)) dalam ruang 3D dapat dinyatakan dengan rumus Euclidean sebagai berikut.

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

Pada tugas kecil II IF2211 Strategi Algoritma kali ini, kami akan mencari pasangan titik yang memiliki jarak terdekat dari kumpulan titik (titik-titiknya dibangkitkan secara acak) menggunakan algoritma *Divide and Conquer*. Kami juga akan memberikan perbandingan antara penggunaan algoritma tersebut dan algoritma *Brute Force* dalam penyelesaian permasalahan ini. Hal yang kami bandingkan adalah banyak perhitungan *Euclidean distance* yang dilakukan serta waktu eksekusi untuk pencarian pasangan titik terdekat tersebut. Khusus untuk titik-titik dalam ruang 3D, kami akan menyajikan *point plotter* untuk menggambarkan semua titiknya dengan pasangan titik yang jaraknya terdekat diberikan warna yang berbeda dari titik-titik lainnya. Selain itu, kami juga akan mengeneralisasi program kami sehingga program kami dapat digunakan untuk mencari sepasang titik terdekat untuk sekumpulan vektor di R^n.

2. Algoritma *Divide and Conquer* untuk Pencarian Pasangan Titik Terdekat 3D

Nama algoritma *Divide and Conquer* berasal dari dua kata, yakni *divide* dan *conquer*. *Divide* berarti persoalan yang besar dibagi menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula, tetapi ukurannya lebih kecil, sedangkan *Conquer (solve)* berarti upa-persoalan masing-masing diselesaikan (diselesaikan secara rekursif jika masih berukuran besar atau diselesaikan secara langsung jika sudah berukuran kecil). Dengan algoritma ini, nantinya solusi masing-masing upa-persoalan akan digabung untuk membentuk solusi persoalan semula.

Divide and Conquer dapat digunakan sebagai algoritma untuk menyelesaikan pencarian titik terdekat dari sekumpulan titik pada dimensi *n*. Permasalahan titik terdekat (*closest pair*) klasik menyelesaikan permasalahan pada 2D dengan algoritma sebagai berikut:

1. Urutkan titik-titik pada himpunan titik S berdasarkan koordinat x. Kompleksitas tahap ini adalah O(n log n) dengan algoritma *sorting merge sort*.

2. **Tahap *divide***: Partisi himpunan titik S menjadi S_1 dan S_2 dengan sebuah garis vertikal *l* yang diambil pada median koordinat x di S.
3. Hitung jarak terdekat pada S_1 dan S_2 sehinga didapatkan δ_1 dan δ_2. Pilih minimum antara keduanya sebagai jarak terdekat δ.
4. **Tahap *conquer*:** Hitung jarak antar titik yang jarak terhadap garis *l*-nya lebih kecil dari δ.

   Tahap ini dapat dilakukan dengan *brute force* secara naif dimana kasus terburuk (dimana terdapat n/2 titik pada masing-masing partisi yang perlu dicek) memiliki $n^2/4$ tahap. Namun, dapat dilakukan optimasi. Tinjau sebuah titik p S_1. Semua titik pada S_2 dalam jarak δ dengan p pasti berada pada sebuah persegi panjang R dengan ukuran δ x 2 δ. Akibatnya, hanya terdapat maksimal 6 titik lainnya yang perlu diperiksa jaraknya dengan titik p sehingga hanya perlu dilakukan 6 x n/2 perhitungan. 6 titik lain ini dapat dengan mudah dipilih dengan melakukan pengurutan himpunan titik berdasarkan koordinat y.

Banyaknya perhitungan dari pemecahan masalah pasangan titik terdekat dengan *Divide and Conquer* adalah T(n) = 2T(n=2) + O(n) = O(n log n).

Meninjau pemecahan masalah pasangan ttitik terdekat pada dimensi dua, algoritma *Divide and Conquer* dapat dikembangkan untuk menyelesaikan permasalahan pada dimensi tiga atau lebih (dimensi d). Misalnya, diberikan himpunan titik S pada dimensi n. Permasalahan ini diselesaikan dengan tahapan sebagai berikut:

1. Urutkan titik-titik pada himpunan titik S berdasarkan koordinat x. Kompleksitas tahap ini adalah O(n log n) dengan algoritma *sorting merge sort*.
2. **Tahap *divide***: Partisi himpunan titik S menjadi S_1 dan S_2 dengan sebuah garis vertikal *l* yang diambil pada median koordinat salah satu sumbu (misalnya sumbu pertama x) di S.
3. Hitung jarak terdekat pada S_1 dan S_2 sehinga didapat kan δ_1 dan δ_2. Pilih minimum antara keduanya sebagai jarak terdekat δ.
4. Proyeksi semua titik yang memiliki jarak lebih kecil dari δ dengan bidang pembagi sebagai himpunan S'.
5. **Tahap *conquer*:** pecahkan permasalahan pasangan titik terdekat dalam S' dengan algoritma *Divide and Conquer* dalam dimensi d – 1 sehingga didapat jarak titik terdekat

dalam S’ yaitu δ’. Misalnya n = 3, maka pasangan titik terdekat yang berada dalam S’ dicari dengan *Divide and Conquer* dalam dimensi 2.

6. Jarak titik terdekat δ = min(δ, δ’).

Banyaknya perhitungan dari pemecahan masalah pasangan titik terdekat dengan *Divide and Conquer* adalah $T(n, d) = 2T(n/2, d) + T(n, d-1) + O(n) = O(n (\log n)^{d-1})$.

3. *Source Code*

3.1 *File* lib/Point.py

```python
import math

class Point:
    dimension : int # dimension
    coordinates : list # list of coordinates in each dimension. coordinates[0] -> dimension 0 (e.g x)
    calculation_count = 0
    """
    Construct an object in Point class

    Args:
        dimension : the dimension of the point

    """
    def __init__(self, dimension: int):
        self.dimension = dimension
        self.coordinates = []

    def get_coordinates(self):
        """
        Inputs coordinates
        """
        for i in range (self.dimension):
            temp = int(input("Enter the coordinate on the point in dimension "+str(i)+ ": "))
            self.coordinates.append(temp)

    def distance(self, other):
        """
        Find the distance between two points

        Args:
            other: the other point

        """
        if self.dimension != other.dimension:
            raise ValueError("Points are not of the same dimension")
        Point.calculation_count += 1
        # Calculate the sum of squared differences in each dimension
        return math.sqrt(sum([(self.coordinates[i] - other.coordinates[i])**2 for i in range(self.dimension)]))
```

```python
def print(self):
    """
    Prints coordinates of the point
    """
    print("(",end="")
    for i in range (self.dimension):
        print(self.coordinates[i],end="")
        if (i!= self.dimension - 1 and not(i==0 and self.dimension ==1)):
            print(", ",end="")
    print(")",end="")
```

3.2 *File* lib/ClosestPair.py

```python
from .Point import *

def brute_force(point_list):
    """
    Recursively find the closest pair of points in point_list using brute force algorithm

    Args:
        point_list (list): List of Point objects

    Returns:
        tuple: A tuple of the minimum distance and the related pair of points
    """
    min = float("inf")
    if(len(point_list)==1):
        return (min, None)
    for i in range (len(point_list)):
        for j in range (i+1, len(point_list)):
            dist = Point.distance(point_list[i], point_list[j])
            if (dist < min):
                min = dist
                closest_pair = (point_list[i], point_list[j])
    return (min, closest_pair)
```

```python
def strip_closest(strip, closest_pair):
    """
    Recursively find the closest pair of points in point_list using divide and conquer algorithm

    Args:
        point_list (list): List of Point objects (prereq: size must be >= 2)
        step (int): current step (the pointss will be sorted by "step" dimension)

    Returns:
        tuple: A tuple of the minimum distance and the related pair of points
    """
    # closest_pair = (strip[0], strip[1])
    dim = strip[0].dimension-1 # use the "last" dimension
    strip = sort_points_by_dimension(strip, dim) # sort the points by the "last" dimension
    n = len(strip)
    for i in range (n):
        for j in range (i+1, n):
            if (strip[j].coordinates[dim] - strip[i].coordinates[dim] >= closest_pair[0]):
                break
            dist = Point.distance(strip[i], strip[j])
            if dist < closest_pair[0]:
                closest_pair = (dist, (strip[i], strip[j]))
    return closest_pair
```

```python
def solve(point_list, step):
    """
    Recursively find the closest pair of points in point_list using divide and conquer algorithm

    Args:
        point_list (list): List of Point objects (prereq: size must be >= 2)
        step (int): current step (the pointss will be sorted by "step" dimension)

    Returns:
        tuple: A tuple of the minimum distance and the related pair of points
    """

    point_list = sort_points_by_dimension(point_list, step)
    n = len(point_list)

    # if the size of the point_list is <= 3, no need to divide
    if (n<=3):
        return brute_force(point_list)

    # get the middle point
    mid = n//2
    mid_point = point_list[mid]

    # partition the list
    left_points = point_list[:mid]
    right_points = point_list[mid:]

    # find the closest pair of each of the partitions
    closest_left_points = solve(left_points, step)
    closest_right_points = solve(right_points, step)

    # find the closer pair between the left closest pair and the right closest pair
    if (closest_left_points[0]<closest_right_points[0]):
        closest_pair = closest_left_points
    else:
        closest_pair = closest_right_points
```

```
    # Handle the points whose distance with the middle slab is smaller than the closest pair distance
    slab = []
    for point in point_list:
        if (abs(point.coordinates[step]-mid_point.coordinates[step]) < closest_pair[0]):
            slab.append(point)

    if (step < mid_point.dimension - 2): # If the current step is examining in larger than 2 dimensional
        closest_pair_slab = solve(slab, step+1)
        if(closest_pair_slab[0] < closest_pair[0]):
            return closest_pair_slab
    else: # The current step is examining in 2D and there are at least two points in the strip
        closest_pair = strip_closest(slab, closest_pair)
    return closest_pair
```

```
def sort_points_by_dimension(point_list, n):
    """
    Sort a list of points by the n-th dimension using merge sort algorithm

    Args:
        point_list (list): List of Point objects
        n (int): Index of the dimension to sort by (0-based)

    Returns:
        list: Sorted list of Point objects
    """
    def merge(left, right):
        result = []
        i = j = 0
        while i < len(left) and j < len(right):
            if left[i].coordinates[n] <= right[j].coordinates[n]:
                result.append(left[i])
                i += 1
            else:
                result.append(right[j])
                j += 1
        result += left[i:]
        result += right[j:]
        return result

    def merge_sort(point_list):
        if len(point_list) <= 1:
            return point_list
        mid = len(point_list) // 2
        left = merge_sort(point_list[:mid])
        right = merge_sort(point_list[mid:])
        return merge(left, right)

    return merge_sort(point_list)
```

3.3 *File* lib/Util.py

```
from .Point import *
from mpl_toolkits.mplot3d import Axes3D
from tkinter import ttk
import tkinter as tk
import matplotlib.pyplot as plt
import numpy as np
import random
import platform
import psutil

def generate_random_points(n_dimension):
    temp_point = Point(dimension = n_dimension)
    for j in range (temp_point.dimension):
        temp = random.uniform(-10, 10)
        temp_point.coordinates.append(temp)
    return temp_point

def splash_screen():
    root = tk.Tk()
    root.geometry("300x200")
    root.overrideredirect(True)
    root.configure(bg = "white")
    text_label = ttk.Label(root, text = "WELCOME! :)", foreground = "black", font = ("Georgia", 24))
    text_label.pack(pady = 65)

    root.update_idletasks()
    width = root.winfo_width()
    height = root.winfo_height()
    x = (root.winfo_screenwidth() // 2) - (width // 2)
    y = (root.winfo_screenheight() // 2) - (height // 2)
    root.geometry("{}x{}+{}+{}".format(width, height, x, y))
    root.after(2000, root.destroy)
    root.mainloop()
```

```python
def show_plotter(point_list, point_1, point_2):
    """
    prereq: dimension must be 3
    """
    figure = plt.figure()
    ax = figure.add_subplot(111, projection='3d')
    x = [point[0] for point in point_list]
    y = [point[1] for point in point_list]
    z = [point[2] for point in point_list]

    colors = ['red' if (point == point_1 or point == point_2) else 'blue' for point in point_list]
    ax.scatter(x, y, z, c = colors)

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    plt.show()
```

```python
def get_num_of_points():
    """
    asks for the num of points and validates if it's an integer
    """
    while True:
        try:
            num_of_points = int(input("Enter the number of points: "))
            if num_of_points > 1:
                return num_of_points
            print("The number of points must be larger than 1.")
        except ValueError:
            print("Invalid input. Please enter an integer.")

def get_dimension():
    """
    asks for the dimension and validates if it's an integer
    """
    while True:
        try:
            dimension = int(input("Enter the dimension of the points: "))
            return dimension
        except ValueError:
            print("Invalid input. Please enter an integer.")
```

```python
def print_computer_spec():
    print("This program is being run on a computer with this specification:")
    # get the system information
    system = platform.uname()

    # get the CPU information
    cpu = platform.processor()

    # get the memory information
    memory = psutil.virtual_memory()
    print("System: {} {}".format(system.system, system.release))
    print("Node name: {}".format(system.node))
    print("CPU: {}".format(cpu))
    print("Memory: {} MB".format(memory.total / (1024*1024)))
```

3.4 *File* main.py

```python
from lib.Point import Point
from lib.ClosestPair import *
from lib.Util import *
import time

continue_solver = True
splash_screen()
while(continue_solver):
    point_list = []
    Point.calculation_count = 0
    print("========================o========================")
    print()
    print("Welcome to our closest points generator!")
    print()
    number_of_points = get_num_of_points()
    dimension = get_dimension()
    for i in range(number_of_points):
        temp_point = generate_random_points(dimension)
        point_list.append(temp_point)

    # Divide and Conquer
    start_time_1 = time.time()
    result_1 = solve(point_list, 0)
    end_time_1 = time.time()
    elapsed_time_1 = end_time_1 - start_time_1
    print()
    print("Result by divide and conquer:")
    print("Distance:", result_1[0])
    print("Point 1: "), result_1[1][0].print()
    print("\n")
    print("Point 2: "), result_1[1][1].print()
    print("\n")

    print("Calculation Count: ", Point.calculation_count)
    print("Time taken:", elapsed_time_1, "seconds", "\n")
    print()
    print()
```

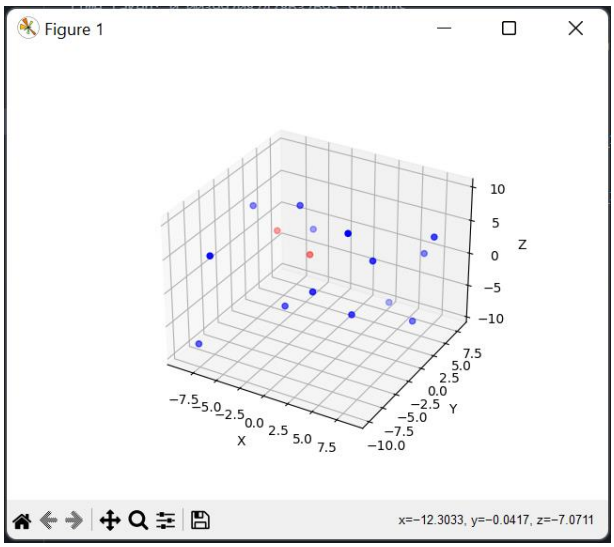
```python
    # Brute Force
    Point.calculation_count = 0
    start_time_2 = time.time()
    result_2 = brute_force(point_list)
    end_time_2 = time.time()
    elapsed_time_2 = end_time_2 - start_time_2

    print("Result by brute force: ")
    print("Distance:", result_2[0])
    print("Point 1: "), result_2[1][0].print()
    print("\n")
    print("Point 2: "), result_2[1][1].print()
    print("\n")

    print("Calculation Count: ", Point.calculation_count)
    print("Time taken:", elapsed_time_2, "seconds", "\n")
    print()
```

```python
    # Point Plotter
    if (dimension == 3):
        list_of_points = [] # List Containing Points (As List)
        for point in point_list:
            list_of_points.append(point.coordinates)
        show_plotter(list_of_points, result_1[1][0].coordinates, result_1[1][1].coordinates)

    valid_continue_choice = False
    while (not valid_continue_choice):
        print("The two closest points have been found! Do you want to try again?")
        continue_choice = input("Input choice (y/n): ")
        if (continue_choice == "y"):
            continue_solver = True
            valid_continue_choice = True
        elif (continue_choice == "n"):
            continue_solver = False
            valid_continue_choice = True
            print()
            print_computer_spec()
            print()
            print("Thank you for trying our closest points generator! :)")
            print()
            print("========================o======================== \n")
        else:
            print("Input invalid! Please input again! \n")
```

4. Testing Program

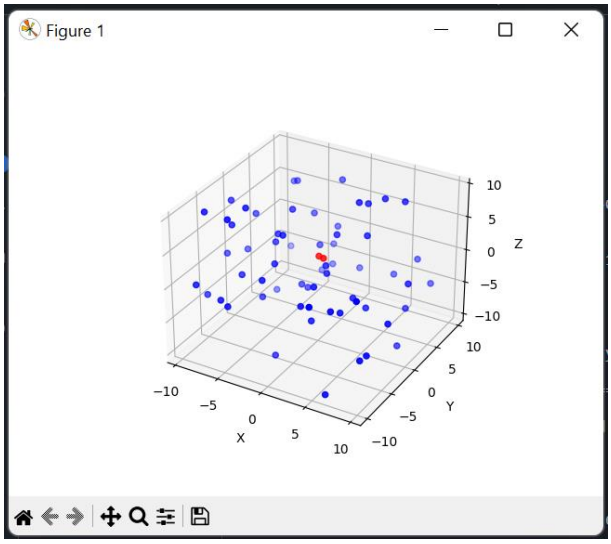
4.1 Titik-titik dalam Ruang 3 Dimensi (16 Titik)


```
=======================O=======================
Welcome to our closest points generator!
Input the number of points to be calculated: 16
Input the dimension: 3
Result by divide and conquer:
Distance: 5.105465776760353
Point 1:
(-2.974077888958555, 2.3188440556420726, -0.5730731959215429)

Point 2:
(-7.6733360394001116, 4.2444671688356586, -0.0489269670935142)

Calculation Count:  59
Time taken: 0.0 seconds

Result by brute force:
Distance: 3.4642746669106383
Point 1:
(-7.6733360394001116, 4.2444671688356586, -0.0489269670935142)

Point 2:
(-4.900837529462734, 6.319957844564502, -0.13132500554780613)

Calculation Count:  120
Time taken: 0.0007827281951904297 seconds

The two closest points have been found! Do you want to try again?
Input choice (y/n): y
```



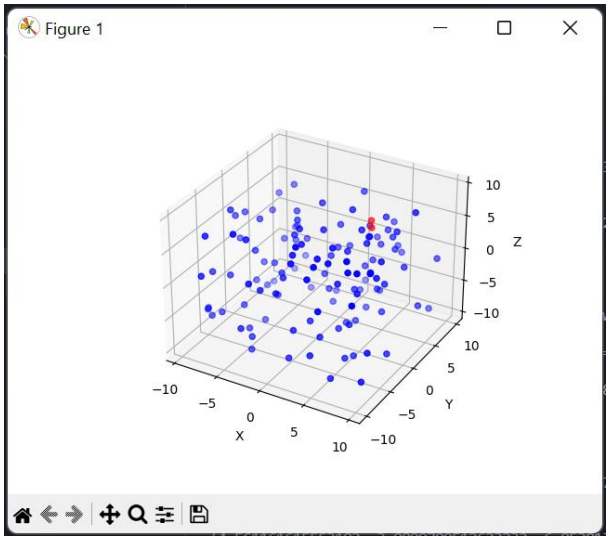
4.2 Titik-titik dalam Ruang 3 Dimensi (64 Titik)

```
Input choice (y/n): y
========================O========================
Welcome to our closest points generator!
Input the number of points to be calculated: 64
Input the dimension: 3
Result by divide and conquer:
Distance: 1.6592780525757371
Point 1:
(1.2938120977912604, -2.056827931556537, 3.6985726556770064)

Point 2:
(2.374449390603889, -3.0834106700902613, 4.427650551437619)

Calculation Count:  219
Time taken: 0.0019996166229248047 seconds

Result by brute force:
Distance: 0.9894871048644024
Point 1:
(-1.862064559307841, 1.2289186911990981, -4.356310839598365)

Point 2:
(-2.246506713176755, 0.6619742293078179, -3.6422639656801508)

Calculation Count:  2016
Time taken: 0.005045652389526367 seconds

The two closest points have been found! Do you want to try again?
Input choice (y/n): y
```



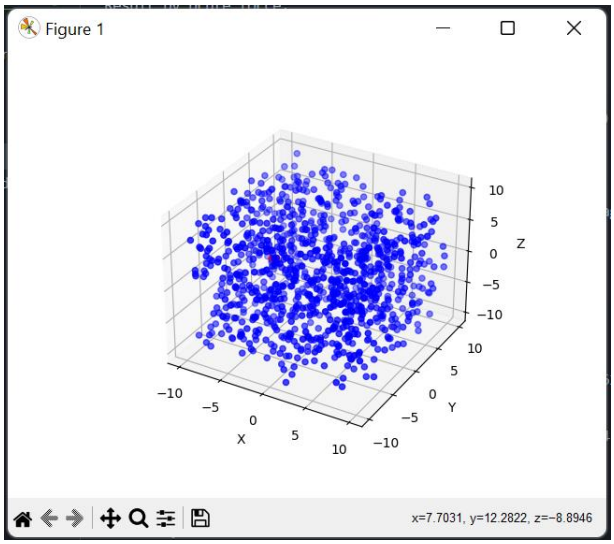
4.3 Titik-titik dalam Ruang 3 Dimensi (128 Titik)

```
Input choice (y/n): y
========================O========================
Welcome to our closest points generator!
Input the number of points to be calculated: 128
Input the dimension: 3
Result by divide and conquer:
Distance: 1.1544051624215175
Point 1:
(5.111599355140594, 2.114447886204033, 6.573684796284159)

Point 2:
(4.561161615553102, 3.0899709512623232, 6.853013139641114)

Calculation Count:  453
Time taken: 0.002999544143676758 seconds

Result by brute force:
Distance: 1.003393127612017
Point 1:
(4.19340209670419, 9.873243353100278, -1.2797130509722017)

Point 2:
(4.641317884784657, 9.889497405416336, -2.177435180091556)

Calculation Count:  8128
Time taken: 0.01423192024230957 seconds

The two closest points have been found! Do you want to try again?
Input choice (y/n): y
```

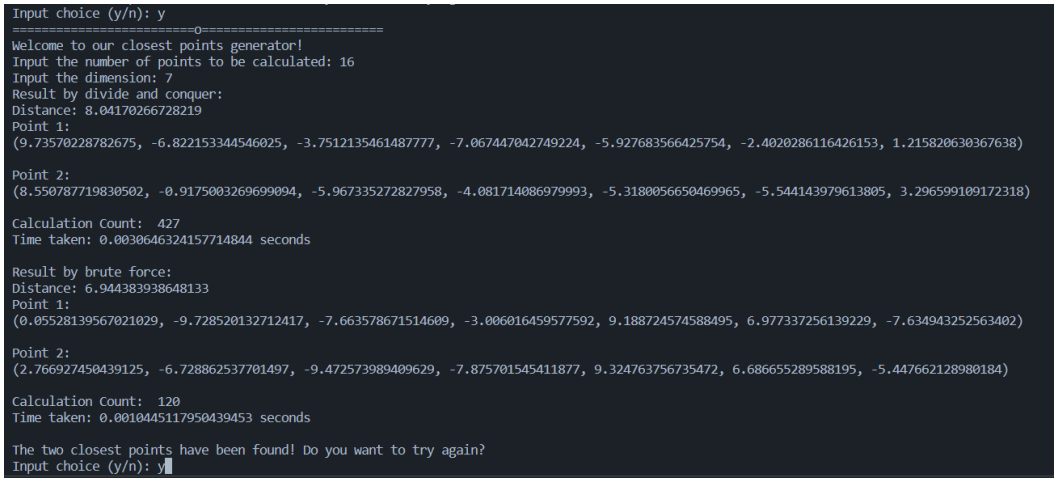


4.4 Titik-titik dalam Ruang 3 Dimensi (1000 Titik)

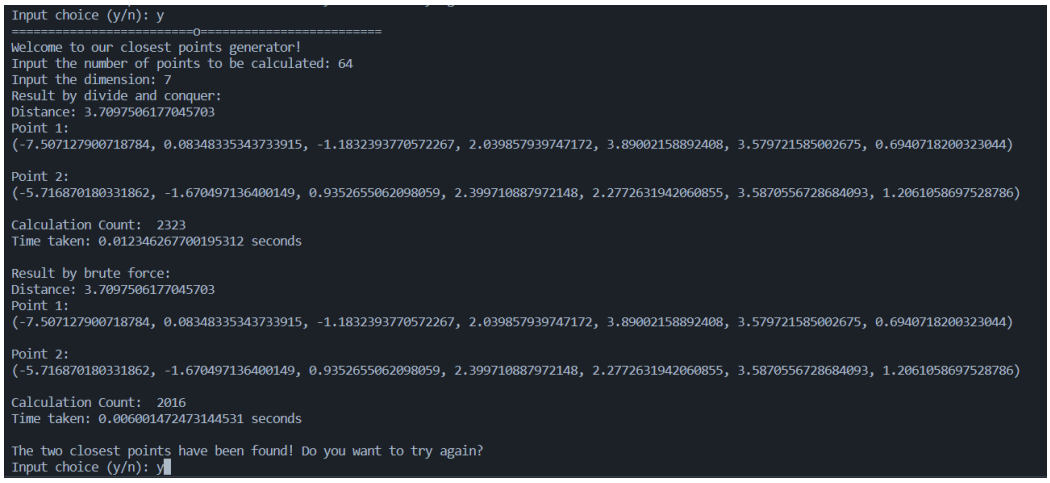
```
Input choice (y/n): y
=======================o=======================
Welcome to our closest points generator!
Input the number of points to be calculated: 1000
Input the dimension: 3
Result by divide and conquer:
Distance: 0.6460250422521351
Point 1:
(-3.8615835158491656, -1.9801842938700442, 1.8463902315720162)

Point 2:
(-4.156351613763243, -2.4960680919101375, 2.1000120575032994)

Calculation Count:  4612
Time taken: 0.018331289291381836 seconds

Result by brute force:
Distance: 0.14412888533000756
Point 1:
(-8.207078614057288, -1.8597003030053116, 9.158100395584949)

Point 2:
(-8.123156462307588, -1.8982063179722886, 9.047431995201038)

Calculation Count:  499500
Time taken: 0.7659962177276611 seconds

The two closest points have been found! Do you want to try again?
Input choice (y/n): y
```



4.5 Titik-titik dalam Ruang N Dimensi (16 Titik)



4.6 Titik-titik dalam Ruang N Dimensi (64 Titik)



4.7 Titik-titik dalam Ruang N Dimensi (128 Titik)

4.8 Titik-titik dalam Ruang N Dimensi (1000 Titik)

```
Input choice (y/n): y
=======================O=======================
Welcome to our closest points generator!
Input the number of points to be calculated: 1000
Input the dimension: 7
Result by divide and conquer:
Distance: 3.4707792866494653
Point 1:
(9.810650885543573, 9.291136023163308, -5.7573284650958545, 4.620215666518789, 8.363794663401041, 5.681901808871423, -0.16799274725743274)

Point 2:
(7.482324718348803, 9.486780514803609, -5.213824159183453, 3.039062089100508, 7.598496778387542, 5.160325839322013, 1.544830488787781)

Calculation Count:  125086
Time taken: 0.46901822090148926 seconds

Result by brute force:
Distance: 1.8401767837805796
Point 1:
(-0.8729917141323682, 7.597614567152824, -5.704186342566981, -8.170732422319094, 1.0171260001509435, -3.200103769206237, -0.6442452950894033)

Point 2:
(-0.8476739825893187, 6.631411350808968, -5.225117405575186, -8.176399739862863, 1.496600723664569, -2.8749493674872166, 0.729399634445528)

Calculation Count:  499500
Time taken: 0.957047700881958 seconds

The two closest points have been found! Do you want to try again?
Input choice (y/n): n
```

```
This program is being run on a computer with this specification:
System: Windows 10
Node name: LAPTOP-VH0J02JC
CPU: Intel64 Family 6 Model 140 Stepping 1, GenuineIntel
Memory: 16179.3046875 MB

Thank you for trying our closest points generator! :)

=======================O=======================
```

5. Pustaka

Rinaldi, M. (2021). Algoritma Divide and Conquer (2021) Bagian 1. Retrieved from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf. (Accessed: February 26, 2023 at 13.41).

6. Kesimpulan

Berdasarkan hasil dari Bab 4, terbukti bahwa algoritma Divide and Conquer jauh lebih efisien untuk memecahkan permasalahan pasangan titik terdekat dibandingkan dengan algoritma Brute Force, terutama untuk jumlah titik yang besar.

7. Lampiran

6.1 *Link Repository*

Berikut adalah tautan *repository* untuk tugas kecil II IF2211 Strategi Algoritma ini.

Link : https://github.com/chaerla/Tucil2_13521044_13521064

6.2 *Checklist*

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