Laporan Tugas Kecil 2

IF2211 - Strategi Algoritma

Mencari Pasangan Titik Terdekat pada Ruang ke-N Menggunakan Algoritma Divide and Conquer

Disusun Oleh:

Muhammad Naufal Nalendra

13521152

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

INSTITUT TEKNOLOGI BANDUNG

2023

BAB I

Deskripsi Masalah

Mencari sepasang titik terdekat dengan Algoritma *Divide and Conquer* sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Pada Tucil 2 kali ini Anda diminta mengembangkan algoritma mencari sepasang titik terdekat pada bidang 3D. Misalkan terdapat *n* buah titik pada ruang 3D. Setiap titik *P* di dalam ruang dinyatakan dengan koordinat *P* = (*x*, *y*, *z*). Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titk *P*1 = (*x*1, *y*1, *z*1) dan *P*2 = (*x*2, *y*2, *z*2) dihitung dengan rumus Euclidean berikut:

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

Buatlah program dalam dalam Bahasa C/C++/Java/Python/Golang/Ruby/Perl (pilih salah satu) untuk mencari sepasang titik yang jaraknya terdekat datu sama lain dengan menerapkan algoritma *divide and conquer* untuk penyelesaiannya, dan perbandingannya dengan Algoritma *Brute Force*.
Masukan program:

- *n*
- titik-titik (dibangkitkan secara acak) dalam koordinat (x, y, z)

Luaran program :

- sepasang titik yang jaraknya terdekat dan nilai jaraknya
- banyaknya operasi perhitungan rumus Euclidian
- waktu riil dalam detik (spesifikasikan komputer yang digunakan)
- Bonus 1 (Nilai = 7,5) penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya.
- Bonus 2 (nilai = 7,5): Generalisasi program anda sehingga dapat mencari sepasang titik terdekat untuk sekumpulan vektor di Rn, setiap vektor dinyatakan dalam bentuk **x** = (x1, x2,…, xn)

BAB II

Dasar Teori

A. Algoritma Divide and Conquer

Divide and conquer adalah suatu teknik dalam pemrograman dan algoritma untuk memecahkan masalah yang sangat rumit. Algoritma bekerja dengan cara membagi permasalahan menjadi subpermasalahan yang lebih kecil dan menyelesaikan seluruh subpermasalahan secara rekursif. Selanjutnya hasil dari semua subpermasalahan digabungkan kembali menjadi solusi masalah secara utuh.

Teknik divide and conquer dapat diterapkan pada berbagai jenis masalah seperti pencarian, pengurutan, perkalian matriks, pencocokan pola, dan banyak lagi. Langkah-langkah umum dalam algoritma divide and conquer adalah sebagai berikut:

1. Divide: Memecahkan masalah menjadi subpermasalahan yang lebih kecil dan sederhana

2. Conquer: Selesaikan submasalah secara terpisah dengan algoritma DnC yang rekursif ataupun iteratif

3. Combine: Menggabungkan seluruh solusi dari subpermasalahan menjadi solusi dari masalah utama

Algoritma *Divide and Conquer* berguna dalam mengurangi kompleksitas waktu dari penyelesaian masalah. Tetapi tidak semua masalah dapat dicari solusi yang lebih baik dengan *Divide and Conquer*. Bahkan pada beberapa kasus seperti permasalahan yang tidak rumit, kompleksitas waktu yang dihasilkan bisa sama atau lebih buruk dari algoritma brute force.

B. Implementasi Algoritma Quicksort

Pada program ini, terdapat tahapan sorting yang dilakukan pada array, untuk menangani hal tersebut, penulis membuat sebuah algoritma quicksort yang menggunakan algoritma divide and conquer. Penjelasan dari pengurutan *quicksort* adalah sebagai berikut :

1. Divide: pilih suatu elemen yang kita sebut *pivot,* pada program penulis memilih elemen terakhir. Selanjutnya bagi *array* menjadi dua sehingga elemen dalam array kiri $\leq$ *pivot* dan yang lainnya selalu $>$ *pivot.*
2. Conquer: ketika *array* hanya memiliki satu elemen, *array* tersebut sudah terurut

3. Combine: Gabungkan kedua array hasil quicksort array yang berisi bagian kiri dan kanan pivot

C. Implementasi Algoritma Closest Pair

Algoritma closest pair berfungsi mencari pasangan titik dengan jarak terdekat pada ruang dua dimensi, tiga dimensi, hingga ruang ke-n. Algoritma closest pair dapat diselesaikan dengan cara mencari jarak antara dua titik dari seluruh titik yang ada menggunakan algoritma *brute force*. Algoritma tersebut memiliki cara yang mudah, tetapi algoritma brute force butuh waktu lebih lama daripada *Divide and Conquer*, karea kompleksitas dari algoritma *brute force* adalah $O(n^2)$, sementara kompleksitas algoritma *Divide and Conquer* pada dimensi-d yaitu $O(n) = n \log^{\mathrm{d}} n$. Oleh karena itu, pada kali ini penulis akan membuat program menggunakan algoritma DnC

1. Divide: Hal pertama yang harus dilakukan adalah mengurutkan titik yang ada sesuai koordinat x. Jika sudah terurut maka bagi kedua himpunan titik dengan sebuah garis tengah yang merupakan median, lalu cari pasangan titik terdekat dari kedua himpunan.
2. Conquer: Saat jumlah titik kurang dari atau sama dengan tiga, pencarian pasangan titik terdekat dapat dilakukan dengan cara mencari jarak antara semua titik yang ada. Jika titik hanya satu, maka program akan menulis ”there is no answer”.
3. Combine: Cari closest pair dari himpunan titik satu dan himpunan titik dua Lalu bandingkan hasil antara hasil himpunan satu dan dua. Ambil nilai yang lebih kecil lalu simpan hasilnya. Selanjutnya dilakukan pemeriksaan pada titik – titik yang berada pada himpunan yang berbeda. Pada tahap ini, titik yang perlu diperiksa hanya titik yang dekat dengan garis median. Lakukan iterasi pada titik-titik yang memenuhi lalu bandingkan dengan hasil pertama. Jika ada nilai yang lebih kecil, maka nilai tersebut disimpan sebagai hasil akhir.

Bab III
Source Code

3.1.File main.py

```
import time
import numpy as np
import platform
from inputs import *
import plotter as pt
import closestPairs as cp
import bruteForce as bf

def main():
    global count
    global rand
    global euclidCount
    # Open the text file in read mode
    with open('src/splashScreen.txt', 'r') as file:
        contents = file.read()
        print(contents)

    # Function to validate inputs
    Input, numPoints, dim = generatePoints()

    distance = 0
    pair = None

    divImpera1 = time.perf_counter()
    # Function to find closest Pair using divide and conquer
    pair, distance = cp.ClosestPair(Input,numPoints)
    dnc = cp.euclidCount
    divImpera2 = time.perf_counter()
    # Function to find closest Pair using brute force
    pairbf, distancebf = bf.bruteForce(Input)
    bfdistance = numpy.round(distancebf, 3)
    bfCount = bf.count
    bruteForceEnd = time.perf_counter()

    # output variables
    divImperaTime = numpy.round(divImpera2 - divImpera1, 8)
    bruteForceTime = numpy.round(bruteForceEnd - divImpera2, 8)
    # Divide and Conquer Algorithm
```

```
    print('-------------------------------------------------------------')
    print('DIVIDE AND CONQUER')
    print('-------------------------------------------------------------')
    print('The closest pair of points are :')
    print(pair[0])
    print(pair[1])
    # print second point
    print('\nWith the distance of :', distance)
    print('And the count of euclidean operations in DnC :', dnc)
    print("Divide and Conquer execution time :",divImperaTime * 1000,"ms")
    # Brute Force Algorithm
    print('-------------------------------------------------------------')
    print('BRUTE FORCE')
    print('-------------------------------------------------------------')
    print('The closest pair of points are :')
    print(pairbf[0])
    print(pairbf[1])
    # print second point
    print('\nWith the distance of :', bfdistance)
    print('And the count of euclidean operations in Brute Force :', bfCount)
    print("Brute Force execution time :",bruteForceTime * 1000,"ms")
    print('-------------------------------------------------------------')
    # PC specifications
    print("\nMachine Specifications : ")
    print("Processor used : ",platform.processor())
    print("System version : ",platform.platform())
    print("Python version : ",platform.python_version())

    visualize = input("\nDo you want to see the plot? (Y/N) ")
    if (visualize == "y" or visualize == "Y"):
        print('-------------------------------------------------------------')
        print("The result will be saved in the folder 'images'")
        name = input("Please enter the desired file name :")
        if(dim == 2):
            pt.plot2d(Input, pair, name)
        elif(dim == 3):
            pt.plot3d(Input, pair, name)
        else:
            print(dim, 'th', 'dimension cannot be plotted')
    else:
        print('Thank you for using our program :) ')

# run main
```

```
main()
```

3.2.File inputs.py

```
import numpy
import os
from sort import quickSort

def validate(prompt, criteria):
    while True:
        try:
            userInput = input(prompt)
            if criteria(userInput):
                return userInput
            else:
                print("Invalid input. Please try again.")
                print('----------------------------------------------------------------')
        except ValueError:
            print("Invalid input. Please try again.")
            print('----------------------------------------------------------------')

def generatePoints():
    random = True

    numPoints = int(validate("\nEnter the number of points: ", lambda x: int(x) > 0))
    dimension = int(validate("Enter the number of dimensions: ", lambda x: int(x) >
0))
    size = int(validate("Enter the max range of a point: ", lambda x: int(x) > 0))

    points = []
    points = numpy.array(points).astype(float)
    Input = setPoints(numPoints, dimension, size, random, points)

    return Input

def setPoints(n_points : int, n_dim : int, size : int, random : bool, inputPoints):
    num = n_points
    dim = n_dim
    max = size
    array = []
    array = numpy.array(array).astype(float)

    if(random == True):
```

```
        array = randomPoints(num,dim,max,array)
    else:
        array = inputPoints
    array = numpy.round(array,3)

    # sort array
    quickSort(array, dim, 0, num-1)
    return array, num, dim

def randomPoints(num : int, dim : int, max : int, array):
    print('----------------------------------------------------------------')
    print("Points are randomized")

    vector = []
    for i in range(num * dim):
        a = numpy.random.uniform(low=0.0, high = max)
        vector.append(a)
    array = numpy.array(vector).reshape(num, dim)
    return array
```

3.3.File bruteForce.py

```
import math
import numpy as np
from closestPairs import euclideanDistance

count= 0

def bruteForce(points : np.array):
    global count
    closest = math.inf
    closestPair = None

    # Iterate over all pairs of points
    for i in range(len(points)):
        for j in range(i+1, len(points)):
            # Calculate the Euclidean distance
            distance = euclideanDistance(points[i], points[j])
            count += 1
            # Check for the closest pair
            if distance < closest:
                closest = distance
                closestPair = (points[i], points[j])
```

```
    return closestPair, closest
```

3.4.File closestPairs.py

```
import numpy

euclidCount = 0

def euclideanDistance(arr1 : numpy.array, arr2 : numpy.array):
    global euclidCount
    euclidCount += 1

    squared = numpy.square(arr1 - arr2)
    squared_sum = numpy.sum(squared)

    return numpy.sqrt(squared_sum)

def ClosestPair(points : numpy.array, numPoints : int):
    closest = float('inf')
    pointsPair = numpy.array([])
    if (numPoints == 1):
        print("There is no answer")
    elif (numPoints == 2):
        closest = euclideanDistance(points[0], points[1])
        pointsPair = numpy.array([0, 1])
    elif (numPoints == 3):
        closest = euclideanDistance(points[1], points[2])
        pointsPair = numpy.array([1, 2])
        for i in range(0, 3):
            if (i != 0):
                temp = euclideanDistance(points[0], points[i])
                if temp < closest:
                    pointsPair = numpy.array([0, i])
                    closest = temp
    else:
        # case 1 : same side
        mid = int(numPoints/2)
        left = points[0:mid]
        right = points[mid:numPoints]

        # split into left and right
        leftPair, leftClosest, = ClosestPair(left, mid)
```

```
        rightPair, rightClosest, = ClosestPair(right, numPoints - mid)

        # select the closest between the left and right subdivision
        if leftClosest < rightClosest:
            closest = leftClosest
            pointsPair = leftPair
        elif leftClosest > rightClosest:
            closest = rightClosest
            pointsPair = mid + rightPair[0:rightPair.size]
        else:
            closest = leftClosest
            pointsPair = numpy.append(leftPair, mid + rightPair[0:rightPair.size])
        # case 2 : diff side
        midPoint = (points[mid - 1][0] + points[mid][0]) / 2
        midPoints = []
        for i in range(numPoints):
            if abs(points[i][0] - midPoint) < closest:
                midPoints.append(points[i])

        # compare each point in the strip with its 7 neighbors (if they exist)
        for i in range(len(midPoints)):
            for j in range(i + 1, min(i + 8, len(midPoints))):
                found = True
                for k in range(len(midPoints[i])):
                    if abs(midPoints[i][k] - midPoints[j][k]) > closest:
                        found = False
                if(found):
                    distance = euclideanDistance(midPoints[i], midPoints[j])
                    if distance < closest:
                        closest = distance
                        id1 = numpy.where((points == midPoints[i]).all(axis=1))[0][0]
                        id2 = numpy.where((points == midPoints[j]).all(axis=1))[0][0]
                        pointsPair = numpy.array([points[id1],points[id2]])

    return (pointsPair, numpy.round(closest, 3))
```

3.5.File sort.py

```
import numpy

def quickSort(points: numpy.array, dimension: int, lowerIdx: int, upperIdx: int):
    if lowerIdx < upperIdx:
        # First partition to search for pivot
```

```
        index = Partition(points, dimension, lowerIdx, upperIdx)
        # Apply recursion for the left and right side of pivot
        quickSort(points, dimension, lowerIdx, index - 1)
        quickSort(points, dimension, index + 1, upperIdx)

def Partition(points: numpy.array, dimension : int, lowerIdx: int, upperIdx: int):
    # Choose first column of last row as pivot
    pivot = points[upperIdx][0]

    # Count the number of pass
    passed = lowerIdx - 1

    # Compare points value with pivot
    for j in range(lowerIdx, upperIdx):
        # Swap if lower than pivot
        if points[j][0] <= pivot:
            passed += 1
            for col in range(dimension):
                temp = points[j][col]
                points[j][col] = points[passed][col]
                points[passed][col] = temp

    # Swap with pivot
    for p in range(dimension):
        temp = points[upperIdx][p]
        points[upperIdx][p] = points[passed+1][p]
        points[passed+1][p] = temp

    # Return index for the next sort
    return (passed+1)
```

3.6.File plotter.py

```
import matplotlib.pyplot as plt
import numpy

def plot2d(pointArr: numpy.array, pairOfPoints : numpy.array, filename : str):
    print('------------------------------------------------------------------')
    print("Please wait a moment..")
    # Create arrays for points and result points
    xres = []
```

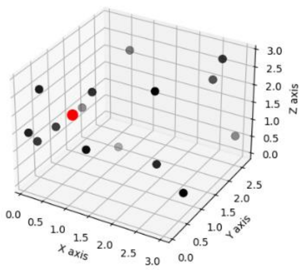
```
    yres = []
    x = []
    y = []

    # Insert points into respective arrays
    numPoints = pointArr.shape[0]
    for i in range(numPoints):
        if i not in pairOfPoints:
            x.append(pointArr[i,0])
            y.append(pointArr[i,1])
        else:
            xres.append(pointArr[i,0])
            yres.append(pointArr[i,1])

    plt.scatter(x, y, color= "black", linewidth=0.5)
    plt.scatter(xres, yres, color="orange", linewidth=1.0)
    plt.savefig('images/' + filename + '.png')

    plt.show()

def plot3d(pointArr: numpy.array, pairOfPoints : numpy.array, filename : str):
    print('------------------------------------------------------------------')
    print("Please wait a moment..")

    fig = plt.figure()
    axis = plt.axes(projection='3d')

    # Create arrays for points and result points
    xres = []
    yres = []
    zres = []
    x = []
    y = []
    z = []

    # Insert points into respective arrays
    numPoints = pointArr.shape[0]
    for i in range(numPoints):
        if i not in pairOfPoints:
            x.append(pointArr[i,0])
            y.append(pointArr[i,1])
            z.append(pointArr[i,2])
        else:
```

```
        xres.append(pointArr[i,0])
        yres.append(pointArr[i,1])
        zres.append(pointArr[i,2])

    # Labeling the axis
    axis.set_xlabel('X axis')
    axis.set_ylabel('Y axis')
    axis.set_zlabel('Z axis')

    axis.scatter(x, y, z, s = 60 ,color= "black", linewidth=0.5)
    axis.scatter(xres, yres, zres, s = 80 ,color="red", linewidth=2.0)
    plt.savefig('images/' + filename + '.png')

    plt.show()
```

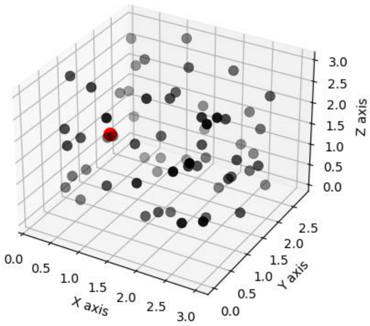
Bab IV
Eksperimen

Pada hasil pengujian, baik algoritma DnC maupun algoritma brute force menghasilkan solusi yang sama, tetapi terkadang terjadi offset antara hasil algoritma DnC dengan brute force. Nilai offset yang terjadi selalu konsisten antara seluruh dimensi titik.

4.1. N = 16

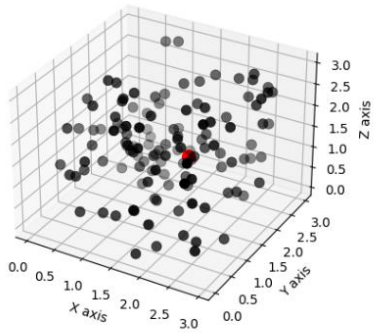


4.2. N = 64




```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

    Mencari pasangan titik terdekat dalam suatu ruang

                       Created by:
            -------------------------------
            | Name : Muhammad Naufal Nalendra |
            | NIM  : 13521152                 |
            -------------------------------

Enter the number of points: 64
Enter the number of dimensions: 3
Enter the max range of a point: 3
---------------------------------------------------------------
Points are randomized
---------------------------------------------------------------
DIVIDE AND CONQUER
---------------------------------------------------------------
The closest pair of points are :
[0.346 2.049 0.709]
[0.37  2.06  0.726]

With the distance of : 0.031
And the count of euclidean operations in DnC : 77
Divide and Conquer execution time : 3.8364 ms
---------------------------------------------------------------
BRUTE FORCE
---------------------------------------------------------------
The closest pair of points are :
[0.346 2.049 0.709]
[0.37  2.06  0.726]

With the distance of : 0.031
And the count of euclidean operations in Brute Force : 2016
Brute Force execution time : 12.265699999999999 ms
---------------------------------------------------------------

Machine Specifications :
Processor used :  Intel64 Family 6 Model 141 Stepping 1, GenuineIntel
System version :  Windows-10-10.0.22000-SP0
Python version :  3.10.8
```

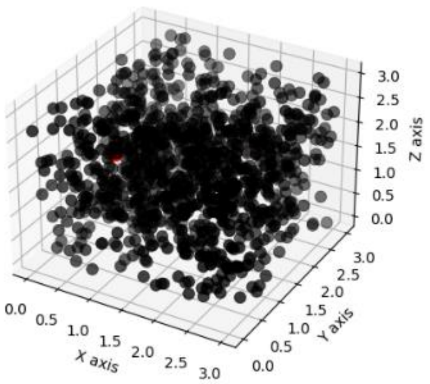
4.3. N = 128




```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

\__//_/ \__//___/\__//___/\__/ /_/    \_,_//_//_/  /___/

    Mencari pasangan titik terdekat dalam suatu ruang
                    Created by:
            -----------------------------------
            | Name : Muhammad Naufal Nalendra |
            | NIM  : 13521152                 |
            -----------------------------------

Enter the number of points: 128
Enter the number of dimensions: 3
Enter the max range of a point: 3
----------------------------------------------------------------------
Points are randomized
----------------------------------------------------------------------
DIVIDE AND CONQUER
----------------------------------------------------------------------
The closest pair of points are :
[16.535 16.156 16.27 ]
[16.581 16.266 16.234]

With the distance of : 0.125
And the count of euclidean operations in DnC : 163
Divide and Conquer execution time : 10.768099999999999 ms
----------------------------------------------------------------------
BRUTE FORCE
----------------------------------------------------------------------
The closest pair of points are :
[0.535 0.156 0.27 ]
[0.581 0.266 0.234]

With the distance of : 0.125
And the count of euclidean operations in Brute Force : 8128
Brute Force execution time : 38.2555 ms
----------------------------------------------------------------------

Machine Specifications :
Processor used :  Intel64 Family 6 Model 141 Stepping 1, GenuineIntel
System version :  Windows-10-10.0.22000-SP0
Python version :  3.10.8
```

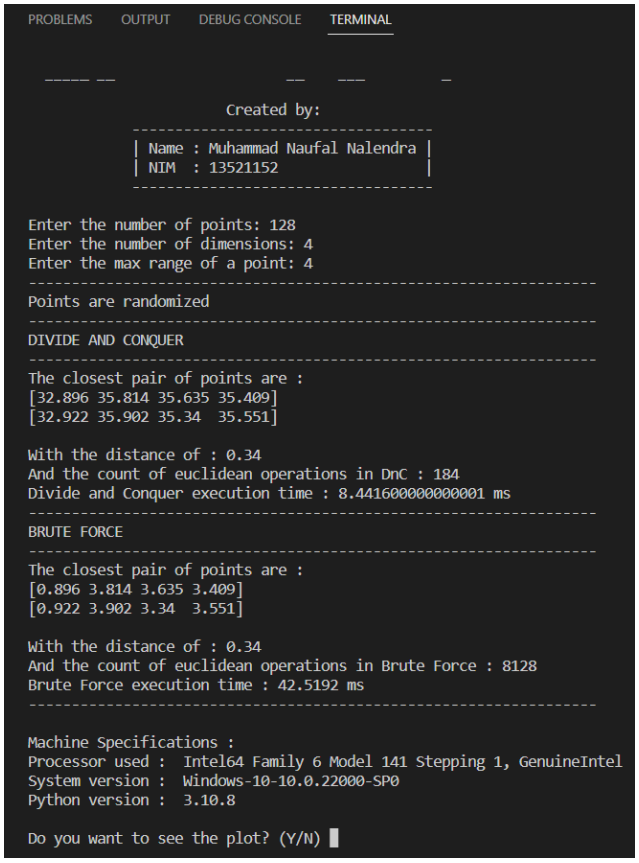
\

4.4. N = 1000

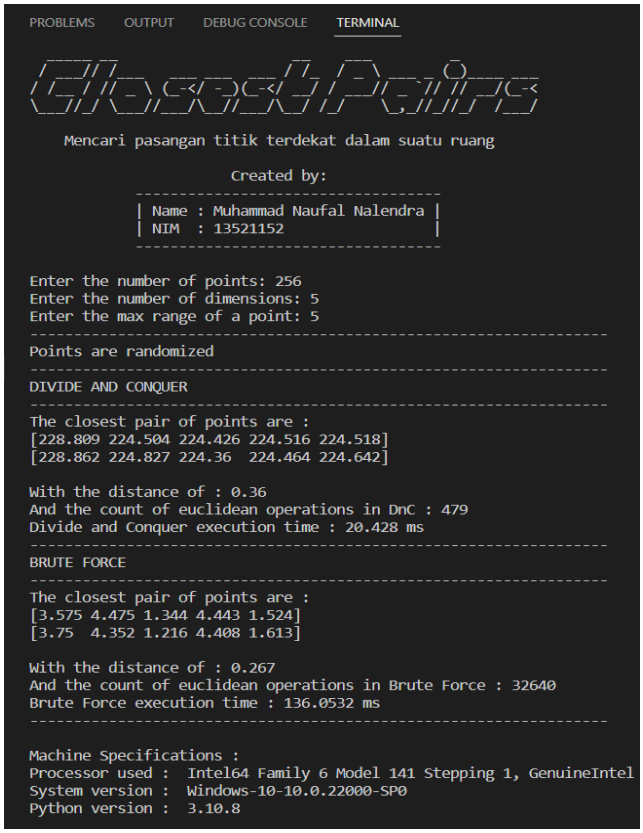



```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

    Mencari pasangan titik terdekat dalam suatu ruang

                        Created by:
             ----------------------------------
             | Name : Muhammad Naufal Nalendra |
             | NIM  : 13521152                 |
             ----------------------------------

Enter the number of points: 1000
Enter the number of dimensions: 3
Enter the max range of a point: 3
-------------------------------------------------------------------
Points are randomized
-------------------------------------------------------------------
DIVIDE AND CONQUER
-------------------------------------------------------------------
The closest pair of points are :
[532.658 533.4   531.997]
[532.674 533.392 531.983]

With the distance of : 0.023
And the count of euclidean operations in DnC : 1323
Divide and Conquer execution time : 37.7966 ms
-------------------------------------------------------------------
BRUTE FORCE
-------------------------------------------------------------------
The closest pair of points are :
[1.658 2.4   0.997]
[1.674 2.392 0.983]

With the distance of : 0.023
And the count of euclidean operations in Brute Force : 499500
Brute Force execution time : 1937.5366000000001 ms
-------------------------------------------------------------------

Machine Specifications :
Processor used :  Intel64 Family 6 Model 141 Stepping 1, GenuineIntel
System version :  Windows-10-10.0.22000-SP0
Python version :  3.10.8
```

4.5. N = 128, dimension = 4



4.6. N = 256, dimension = 5



Link Repo :

https://github.com/Naufal-Nalendra/Tucil2_13521152

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

Tabel 4.1 Penilaian