

Laporan Tugas Kecil 2 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

**Pencarian Pasangan Titik Terdekat di Bidang 3D
dengan Algoritma Divide and Conquer**



Hobert Anthony Jonatan
13521079

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

2023

DAFTAR ISI

BAB 1 : DESKRIPSI MASALAH	2
BAB 2 : PENERAPAN ALGORITMA DIVIDE AND CONQUER	3
2.1 Penerapan Algoritma Divide and Conquer untuk Menyelesaikan persoalan	3
2.2 Analisis Efisiensi Algoritma	4
BAB 3 : IMPLEMENTASI PROGRAM DENGAN PYTHON	5
BAB 4 : EKSPERIMEN	22
LAMPIRAN	26

BAB 1 : 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 Tugasil 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 titik $P_1 = (x_1, y_1, z_1)$ dan $P_2 = (x_2, y_2, z_2)$ dihitung dengan rumus Euclidean berikut:

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

Buatlah program dalam Bahasa C/C++/Java/Python/Golang/Ruby/Perl (pilih salah satu) untuk mencari pasangan titik yang jaraknya terdekat satu sama lain dengan menerapkan algoritma Divide and Conquer untuk penyelesaiannya, dan perbandingannya dengan Algoritma Brute Force.

Masukkan 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 Euclidean
- 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 R^n , setiap vektor dinyatakan dalam bentuk $x = (x_1, x_2, \dots, x_n)$.

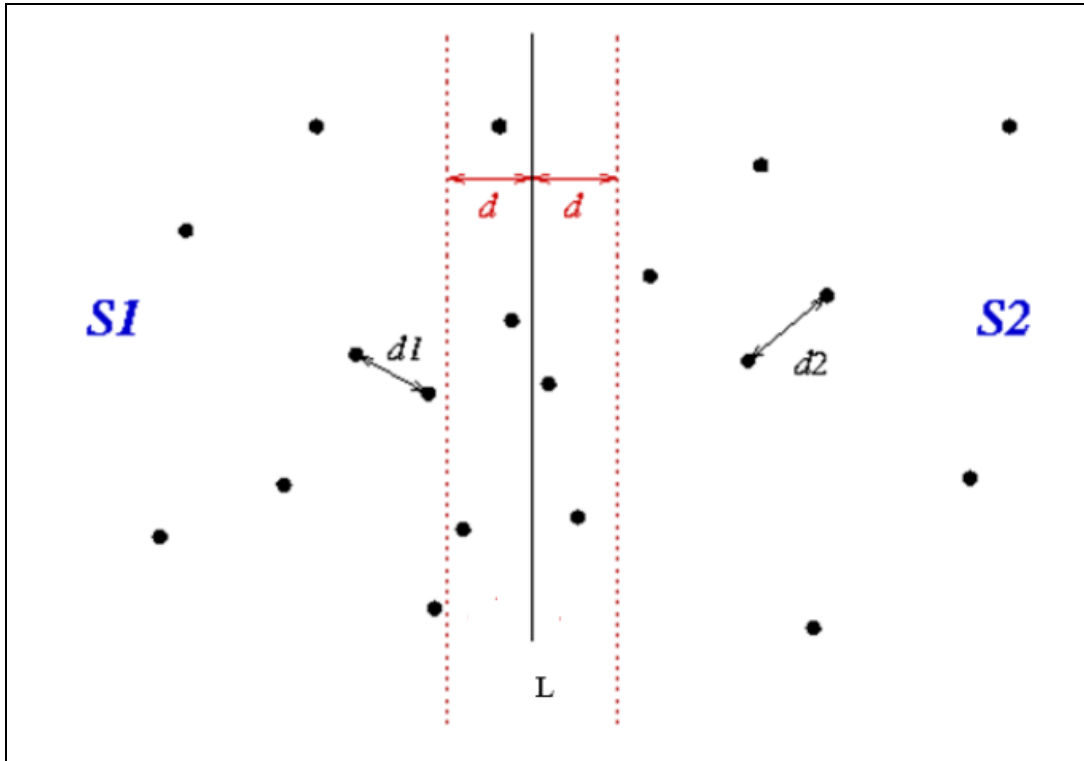
BAB 2 : PENERAPAN ALGORITMA DIVIDE AND CONQUER

Divide and Conquer adalah sebuah pendekatan penyelesaian masalah dengan cara Divide, Conquer, kemudian diikuti dengan Combine. Divide berarti membagi persoalan menjadi beberapa upa-persoalan atau sub persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya berukuran hampir sama). Selanjutnya Conquer (*solve*) berarti menyelesaikan masing-masing upa-persoalan, dapat diselesaikan secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar. Tahap terakhir adalah Combine, yaitu menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi untuk persoalan semula

2.1 Penerapan Algoritma Divide and Conquer untuk Menyelesaikan persoalan

Penyelesaian permasalahan pasangan titik terdekat di 3D diselesaikan dengan algoritma Divide and Conquer dengan mengikuti langkah-langkah berikut :

1. Urutkan kumpulan titik berdasarkan koordinat x (terurut membesar, pada implementasi dalam program ini menggunakan algoritma *quick sort*)
2. Bagi kumpulan titik menjadi dua bagian dengan acuan adalah koordinat x median, akan ada himpunan bagian kiri dan kanan terhadap koordinat x median tersebut (ini adalah tahap *Divide* pada algoritma ini)
3. Secara rekursif, temukan pasangan titik terdekat di setiap bagian dengan menggunakan pendekatan pembagian kumpulan titik menjadi dua bagian secara terus menerus
4. Setelah menemukan jarak terdekat di bagian kiri dan bagian kanan, ambil jarak paling minimum di antara keduanya sebagai minimum global sementara, kemudian pertimbangkan semua titik dalam koordinat jarak sebesar minimum global tadi dari koordinat x median, dan urutkan kumpulan titik tersebut berdasarkan koordinat y nya.
5. Simpan nilai minimum global tadi dalam variabel baru, misalnya dengan nama *mid_closest*
6. Lakukan perhitungan jarak antar titik untuk setiap titik dalam himpunan ini, apabila menemukan pasangan titik dengan jarak lebih pendek dibanding *mid_closest*, maka perbaharui nilai *mid_closest* dengan jarak pasangan titik tersebut.
7. Kembalikan nilai terkecil antara *mid_closest* dan minimum global sebelumnya sebagai jarak terdekat di antara sekumpulan titik tersebut.



Ilustrasi algoritma *Divide and Conquer* pada pencarian pasangan titik terdekat di 2D
sumber :

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf)

2.2 Analisis Efisiensi Algoritma

Algoritma *Brute Force* juga diterapkan dalam mencari pasangan titik terdekat pada Tugas Kecil kali ini, algoritma *brute force* memiliki kompleksitas waktu $O(n^2)$ karena untuk setiap titik perlu dicari jarak ke seluruh titik lainnya untuk kemudian dicari mana jarak yang paling minimum, sedangkan untuk algoritma *Divide and Conquer* dalam mencari pasangan titik terdekat hanya memiliki kompleksitas $O(n \log n)$ karena algoritma ini secara rekursif terus menerus membagi kumpulan titik menjadi dua bagian sama besar (menyebabkan kompleksitas iterasi pertama $O(\log n)$) dan kemudian untuk setiap iterasi di dalam rekursif akan ditinjau sejumlah n titik sehingga menyebabkan kompleksitas total algoritma menjadi $O(n \log n)$ yang mana memiliki efisiensi yang jauh lebih baik dibanding algoritma *brute force* terutama untuk n besar.

BAB 3 : IMPLEMENTASI PROGRAM DENGAN PYTHON

Algoritma yang telah dijabarkan pada Bab 2 diimplementasikan dengan program dalam bahasa Python. Source code dari program ini dapat dilihat pada tautan repository GitHub yang telah dilampirkan pada bagian Lampiran. Source code program berada pada folder src, yang berisikan 6 file, yaitu **main.py**, **closestPair3D.py**, **closestPairND.py**, **generatePoints.py**, **IO.py**, dan **sort.py**

Berikut adalah *source code* dari program yang telah dibuat :

File main.py

File ini berisi program utama dan juga algoritma *Divide and Conquer* untuk dimensi 3 Dimensi dan juga N dimensi. berikut adalah *source code* dalam file main.py

```
# file : main.py
# berisi program utama

# import module
from generatePoints import *
from IO import *
from closestPair3D import *
from closestPairND import *
from generatePoints import *

# fungsi Divide and Conquer untuk mencari sepasang titik terdekat
# di 3D
def DnCclosestPair3D(listRandomDots):
    global countSteps

    # pengurutan titik berdasarkan koordinat x nya
    listRandomDots = quickSort(listRandomDots,0)
    # kasus basis, bisa diselesaikan dengan bruteForce karena n
    bernilai cukup kecil
    n = len(listRandomDots)
    if n <= 3:
        m = bruteForceClosestPair3D(listRandomDots)
        countSteps += m[3]
        return (m[1],m[0])
```

```

    # cari titik tengah untuk membagi kumpulan titik menjadi dua
    bagian
    # dengan jumlah yang sama banyak
    mid = n // 2

    # bagi titik menjadi 2 himpunan, yaitu kiri dan kanan relatif
    terhadap titik tengah
    leftPoints = listRandomDots[:mid]
    rightPoints = listRandomDots[mid:]

    # cari closest pair pada kedua bagian himpunan secara rekursif
    leftClosest = DnCclosestPair3D(leftPoints)
    rightClosest = DnCclosestPair3D(rightPoints)

    # ambil pasangan titik dengan jarak terdekat di antara kiri
    atau kanan
    if min(leftClosest[0], rightClosest[0]) == leftClosest[0]:
        closestPair = leftClosest[1]
    else:
        closestPair = rightClosest[1]

    closest = min(leftClosest[0], rightClosest[0])

    # lakukan pengecekan lagi untuk titik-titik di sekitar garis
    bagi
    midPoints = []
    for dots in listRandomDots:
        if abs(dots[0] - listRandomDots[mid][0]) < closest:
            midPoints.append(dots)

    # pengurutan titik midpoint berdasarkan nilai sumbu y nya
    midPoints = quickSort(midPoints, 1)
    midClosest = closest
    for i in range(len(midPoints)):
        for j in range(i + 1, len(midPoints)):
            if midPoints[j][1] - midPoints[i][1] >= midClosest:
                break
            dist = getDistanceBetween3D(midPoints[i], midPoints[j])

```

```

        countSteps += 1
        if dist < midClosest:
            midClosest = dist
            midClosestPair = (midPoints[i], midPoints[j])

    if(min(closest, midClosest) == closest):
        return (closest, closestPair)
    else:
        return (midClosest, midClosestPair)

# fungsi Divide and Conquer untuk mencari sepasang titik terdekat
# di N-D
def DnCclosestPairND(points):
    global countStepN

    if len(points) <= 3:
        m = bruteForceClosestPairND(points)
        countStepN += m[3]
        return (m[1], m[0])

    points = quickSort(points, 0)

    mid = len(points) // 2
    left = points[:mid]
    right = points[mid:]

    leftClosest = DnCclosestPairND(left)
    rightClosest = DnCclosestPairND(right)

    d = min(leftClosest[0], rightClosest[0])

    if(d == leftClosest[0]):
        closestPair = leftClosest[1]
    else:
        closestPair = rightClosest[1]

    mid_x = points[mid][0]

```



```

strip = [p for p in points if abs(p[0] - mid_x) < d]

if(len(points[0]) > 1):
    n = 1
else:
    n = 0
strip = quickSort(strip,n)
for i in range(len(strip)):
    for j in range(i+1, len(strip)):
        if strip[j][n] - strip[i][n] >= d:
            break
        dist = getDistanceBetweenND(strip[i], strip[j])
        countStepN += 1
        if dist < d:
            d = dist
            closestPair = (strip[i],strip[j])

return (d, closestPair)

def main():
    global countStepN
    global countSteps

    run = True
    printWelcomeScreen()

    while (run):
        countSteps = 0
        countStepN = 0
        printMenu()
        inputValid = False

        while(not inputValid):
            try:
                choice = int(input("Masukkan pilihan anda (1-3): "))
                if(1 <= choice <= 3):
                    inputValid = True

```

```

        print()
        if(choice == 1):
            print("Penyelesaian masalah closest pair pada
3 Dimensi")

            print()
            elif(choice == 2):
                print("Penyelesaian masalah closest pair pada
N Dimensi")

                print()

        else:
            print()
            print("Input tidak valid!, silakan ulangi masukan
Anda!")

            print()
    except:
        print("Masukan Anda salah!")
        print("Silahkan ulangi kembali masukan.")
        print()

if(choice == 1):
    # ambil input n
    n = getAndValidateInputN()

    # generate titik random
    randomPoints = generateRandom3DPoints(n)

    # eksekusi algoritma DNC
    startTime = time.time() * 1000
    hasil_dnc = DnCClosestPair3D(randomPoints)
    endTime = time.time() * 1000
    exeTime = endTime - startTime

    hasil_BF = bruteForceClosestPair3D(randomPoints)

    printHasilBruteForce(hasil_BF)
    printHasilDnC(randomPoints, hasil_dnc, exeTime, countSteps)

```

```

elif(choice == 2):
    # ambil input n
    n = getAndValidateInputN()
    d = getAndValidateInputD()

    # generate titik random
    randomPoints = generatePoint(n,d)

    hasil_BF = bruteForceClosestPairND(randomPoints)

    if(d == 3):
        startTime = time.time() * 1000
        hasil_dnc = DnCclosestPair3D(randomPoints)
        endTime = time.time() * 1000
        exeTime = endTime - startTime
        printHasilBruteForce(hasil_BF)
        printHasilDnC(hasil_dnc,exeTime,countSteps)
    else:
        startTime = time.time() * 1000
        hasil_dnc = DnCclosestPairND(randomPoints)
        endTime = time.time() * 1000
        exeTime = endTime - startTime
        printHasilBruteForce(hasil_BF)
        printHasilDnCND(hasil_dnc,exeTime,countStepN)

    else:
        run = False

printExitScreen()

# main program
if __name__ == '__main__':
    main()

```

File generatePoints.py

file generatePoints.py berisi fungsi untuk menghasilkan titik random di ruang 3D maupun ruang N-D. berikut adalah *source code* dalam file generatePoints.py

```
# file : generatePoints.py
# berfungsi untuk menghasilkan titik 3 dimensi dan juga n-dimensi
# secara random sesuai dengan masukan jumlah titik
# domain dari titik adalah -100 sampai dengan 100
# setiap koordinat sumbu spesifik hingga 3 desimal

import random

# fungsi untuk menggenerate titik sejumlah n dalam ruang 3D
def generateRandom3DPoints(n):
    points = []
    for i in range(n):
        x = round(random.uniform(-100,100),3)
        y = round(random.uniform(-100,100),3)
        z = round(random.uniform(-100,100),3)

        point = [x,y,z]
        points.append(point)

    return points # return list of titik

# fungsi untuk menggenerate titik sejumlah n dalam ruang k-D
def generatePoint(n,k):
    points = []
    for i in range(n):
        point = []
        for i in range(k):
            t = round(random.uniform(-100,100),3)
            point.append(t)
        points.append(point)

    return points
```

File closestPair3D.py

File `closestPair3D.py` berisi fungsi untuk mencari jarak antar titik dalam ruang 3 dimensi dan juga mencari solusi pasangan closest pair dengan algoritma brute force. Berikut adalah *source code* dalam file `closestPair3D.py`

```
# file : closestPair3D.py
# Source code program untuk mencari sepasang titik terdekat di
ruang 3D
# Tersedia algoritma brute force dan juga Divide and Conquer

# import external module
from math import sqrt, pow
import time
from sort import quickSort

# fungsi untuk mendapat jarak antara titik1 dan titik2
def getDistanceBetween3D(p1,p2):
    return sqrt(pow((p1[0]-p2[0]),2) + pow((p1[1]-p2[1]),2) +
pow((p1[2]-p2[2]),2))

# fungsi algoritma brute force untuk mencari closest pair
def bruteForceClosestPair3D(listRandomDots):
    closestPair = ()
    closestPairIndex = ()
    size = len(listRandomDots)
    minDistance = None
    countStep = 0
    initTime = time.time() * 1000
    for i in range(size):
        for j in range(i+1,size):
            p1 = listRandomDots[i]
            p2 = listRandomDots[j]
            distance = getDistanceBetween3D(p1,p2)
            if minDistance is None:
                minDistance = distance
                closestPair = (p1,p2)
                closestPairIndex = (i,j)
```

```

        elif distance < minDistance:
            minDistance = distance
            closestPair = (p1,p2)
            closestPairIndex = (i,j)
            countStep += 1

endTime = time.time() * 1000
exeTime = endTime - initTime

    return (closestPair, minDistance, exeTime, countStep,
closestPairIndex)

```

File closestPairND.py

File closestPairND.py berisi fungsi untuk mencari jarak antar titik dalam ruang N dimensi dan juga mencari solusi pasangan closest pair di ruang N dimensi dengan algoritma brute force. Berikut adalah *source code* dalam file closestPairND.py

```

# file : closestPairND.py
# Source code program untuk mencari sepasang titik terdekat di
ruang 3D
# Tersedia algoritma brute force dan juga Divide and Conquer

import math
import time
from generatePoints import generatePoint
from sort import quickSort

def getDistanceBetweenND(p1, p2):
    return math.sqrt(sum((p1[i] - p2[i])**2 for i in range(len(p1))))

def bruteForceClosestPairND(points):
    closestPair = ()
    minDistance = None
    countStep = 0
    size = len(points)
    initTime = time.time() * 1000
    for i in range(size):
        for j in range(i+1, size):

```

```

        p1 = points[i]
        p2 = points[j]
        distance = getDistanceBetweenND(p1,p2)
        if minDistance is None:
            minDistance = distance
            closestPair = (p1,p2)
        elif distance < minDistance:
            minDistance = distance
            closestPair = (p1,p2)
        countStep += 1
    endTime = time.time() * 1000
    exeTime = endTime - initTime

    return (closestPair, minDistance, exeTime, countStep)

```

File sort.py

File sort.py berisi algoritma quickSort untuk mengurutkan titik berdasarkan sumbu tertentu. Berikut adalah *source code* dalam file sort.py

```

# file : sort.py

# fungsi quick sort untuk mengurutkan titik terurut menaik
# berdasarkan koordinat sebuah sumbu n
# n = 0, maka x
# n = 1, maka y, dst..
def quickSort(points,n):
    if len(points) <= 1:
        return points
    else:
        pivot = points[0][n] # choose x value of first point as
# pivot
        left = [p for p in points[1:] if p[n] <= pivot]
        right = [p for p in points[1:] if p[n] > pivot]
        return quickSort(left,n) + [points[0]] + quickSort(right,n)

```

File IO.py

File IO.py berisi fungsi dan prosedur yang menangani input dan output yang dibutuhkan oleh program ini, sifatnya hanya sebagai pendukung. Berikut adalah *source code* dalam file IO.py

```
# file IO.py
# berfungsi untuk handle input dan output program

# import external module
import pyfiglet
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import time
from sort import quickSort

# prosedur untuk menampilkan menu
def printMenu():
    print("""
    ##### MENU #####
    1. Closest Pair 3 Dimension
    2. Closest Pair N Dimension
    3. Exit
    """)

# prosedur untuk menampilkan welcome text
def printWelcomeScreen():
    welcomeText = pyfiglet.figlet_format("Closest Pair 3D Solver")
    print(welcomeText)
    print()
    print('#' * 80)
    print()

# prosedur untuk menampilkan exit text
def printExitScreen():
    print()
    print('#' * 80)
    print()
    exitText = pyfiglet.figlet_format("THANK YOU! ^-^")
```



```

print(exitText)
print()

# fungsi untuk meminta dan memvalidasi nilai dimensi
def getAndValidateInputD():
    correctInput = False
    while not correctInput:
        try:
            d = int(input("Masukkan dimensi: "))
            print()
        except:
            print("Masukkan Anda tidak sesuai, silakan ulangi masukan
Anda")
            print()

            if(type(d) == int):
                if(d <= 0):
                    print("Dimensi tidak bisa bernilai 0 ataupun kurang
dari 0")
                    print("Silakan ulangi kembali masukan.")
                    print()
                else:
                    correctInput = True

    return d

# prosedur untuk minta dan validasi input n
def getAndValidateInputN():
    correctInput = False
    while not correctInput:
        try:
            n = int(input("Masukkan jumlah titik (n): "))
            print()
        except:
            print("Masukkan Anda tidak sesuai, silahkan ulangi
masukan Anda")

```

```

        print()
        n = "salah"

    if(type(n) == int):
        if(n < 2):
            print("Untuk mencari pasangan titik terdekat
diperlukan minimal 2 titik!")
            print("Silakan ulangi kembali masukan.")
            print()
        else:
            correctInput = True

    return n

def continueOrNot():
    print()
    print('#' * 80)
    print()
    answer = False
    ask = input("Ingin mencoba jumlah titik lainnya? [y/n]:
").lower()
    if(ask == 'y'):
        answer = True
        print()
        print("#" * 80)
        print()

    return answer

# prosedur untuk mengeluarkan output hasil program dari algoritma
divide and conquer
def printHasilDnC(randomPoints, hasil, exeTime, countStep):
    print("Hasil dari run dengan algoritma DIVIDE AND CONQUER : ")
    print()
    print(f"Pasangan titik terdekat                : {hasil[1][0]} ,
{hasil[1][1]}")
    print(f"Jarak pasangan titik terdekat          : {hasil[0]} satuan")

```

```

        print(f"Waktu eksekusi program          : {exeTime}
milisekon")
        print(f"Jumlah perhitungan rumus Euclidean  : {countStep}")

        print()
        print('#' * 80)
        print()
        answer = input("Ingin menampilkan visualisasi titik di ruang 3D?
[y/n]: ").lower()
        if answer == 'y':
            visualize3D(quickSort(randomPoints,0),hasil[1])

# prosedur untuk mengeluarkan output hasil program dari algoritma
divide and conquer
def printHasilDnCND(hasil,exeTime,countStep):
    print("Hasil dari run dengan algoritma DIVIDE AND CONQUER : ")
    print()
    print(f"Pasangan titik terdekat          : {hasil[1][0]} ,
{hasil[1][1]}")
    print(f"Jarak pasangan titik terdekat      : {hasil[0]} satuan")
    print(f"Waktu eksekusi program          : {exeTime}
milisekon")
    print(f"Jumlah perhitungan rumus Euclidean  : {countStep}")

    print()
    print('#' * 80)
    print()

# prosedur untuk mengeluarkan output hasil program dari algoritma
brute force
def printHasilBruteForce(hasil):
    print("#" * 80)
    print()
    print("Hasil dari run dengan algoritma BRUTE FORCE : ")
    print()
    print(f"Pasangan titik terdekat          : {hasil[0][0]} ,
{hasil[0][1]}")

```

```

    print(f"Jarak pasangan titik terdekat      : {hasil[1]} satuan")
    print(f"Waktu eksekusi                      : {hasil[2]}
milisekon")
    print(f"Jumlah perhitungan rumus Euclidian  : {hasil[3]}")
    print()
    print('#' * 80)
    print()

# procedure untuk visualisasi seluruh titik dan 2 titik dengan
jarak terdekat (SPEK BONUS)
def visualize3D(randomPoints,closestPairCoordinate):
    print()
    print("Sepasang titik berwarna biru menunjukkan sepasang titik
terdekat.")
    print()
    print("Menampilkan visualisasi data...")
    time.sleep(0.5)
    print('.')
    time.sleep(0.5)
    print('.')

    # mewarnai seluruh titik dengan warna merah
    colors = np.array(['r'] * len(randomPoints))
    first = closestPairCoordinate[0];
    second = closestPairCoordinate[1];
    pertama = 0
    kedua = 0

    # mencari index titik dalam kumpulan titik
    for i in range(len(randomPoints)):
        case1 = (randomPoints[i][0] == first[0] and
randomPoints[i][1] == first[1] and randomPoints[i][2] == first[2])
        case2 = (randomPoints[i][0] == second[0] and
randomPoints[i][1] == second[1] and randomPoints[i][2] == second[2])
        if(case1 or case2):
            pertama = i
            break

```

```

# mencari index pasangan titik lainnya dalam kumpulan titik
for j in range(pertama+1, len(randomPoints)):

    if(case1):
        # jika menemukan titik pertama terlebih dahulu, berarti
        # yang selanjutnya dicari adalah titik kedua
        toSearch = (randomPoints[j][0] == second[0] and
randomPoints[j][1] == second[1] and randomPoints[j][2] == second[2])
    elif(case2):
        # menemukan titik kedua terlebih dahulu, berarti yang
        # selanjutnya dicari adalah titik pertama
        toSearch = (randomPoints[j][0] == first[0] and
randomPoints[j][1] == first[1] and randomPoints[j][2] == first[2])

    if((toSearch)):
        kedua = j
        break

# mengganti warna sepasang titik terdekat dengan warna biru
colors[pertama] = colors[kedua] = 'b'

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x_dots = []
y_dots = []
z_dots = []

for i in range(len(randomPoints)):
    x_dots.append(randomPoints[i][0])

for i in range(len(randomPoints)):
    y_dots.append(randomPoints[i][1])

for i in range(len(randomPoints)):
    z_dots.append(randomPoints[i][2])

ax.scatter(x_dots, y_dots, z_dots, c = colors)

```

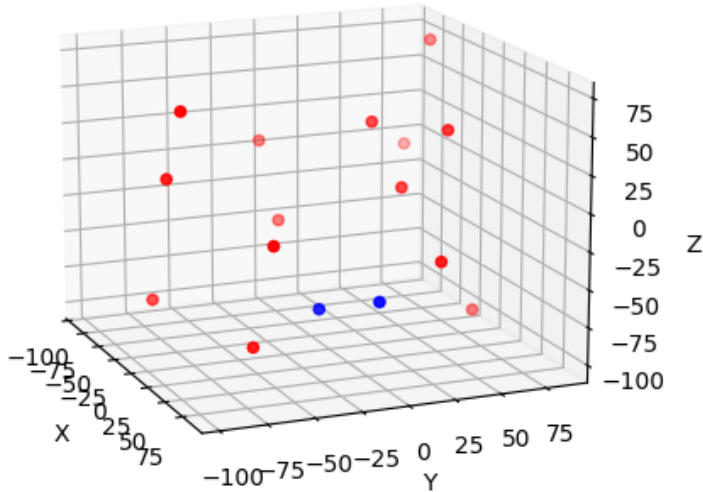
```
ax.set_xlabel('X')  
ax.set_ylabel('Y')  
ax.set_zlabel('Z')  
  
plt.show()
```

Library pendukung yang digunakan :

- random
- matplotlib
- numpy
- math
- time
- pyfiglet

BAB 4 : EKSPERIMEN

Test case program ini dijalankan pada komputer dengan processor AMD Ryzen 9 5900HS with Radeon Graphics, 3301 Mhz, 8 Core(s), 16 Logical Processor(s) , dengan RAM 16GB dan berjalan pada Windows 11. Berikut adalah beberapa contoh kasus saat program dijalankan.

Test Case	Luaran Program
n = 16, dimensi = 3	<pre> Penyelesaian masalah closest pair pada 3 Dimensi Masukkan jumlah titik (n): 16 ##### Hasil dari run dengan algoritma BRUTE FORCE : Pasangan titik terdekat : [58.549, -30.104, -59.201] , [88.794, -10.535, -46.746] Jarak pasangan titik terdekat : 38.11604401036393 satuan Waktu eksekusi : 0.0 milisekon Jumlah perhitungan rumus Euclidian : 120 ##### Hasil dari run dengan algoritma DIVIDE AND CONQUER : Pasangan titik terdekat : [58.549, -30.104, -59.201] , [88.794, -10.535, -46.746] Jarak pasangan titik terdekat : 38.11604401036393 satuan Waktu eksekusi program : 0.0 milisekon Jumlah perhitungan rumus Euclidean : 40 ##### </pre> 

n = 64, dimensi = 3

```
Penyelesaian masalah closest pair pada 3 Dimensi
Masukkan jumlah titik (n): 64

#####

Hasil dari run dengan algoritma BRUTE FORCE :

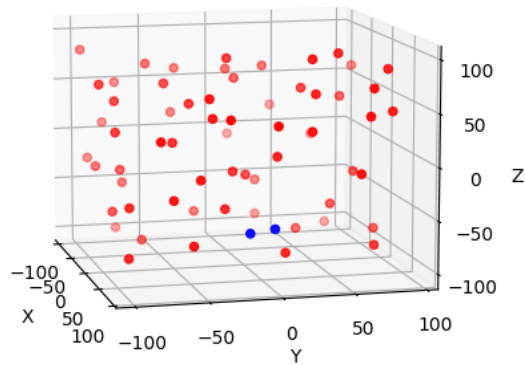
Pasangan titik terdekat      : [83.975, -16.828, -58.878] , [81.702, 0.868, -56.6]
Jarak pasangan titik terdekat : 17.986223311190148 satuan
Waktu eksekusi               : 1.510009765625 milisekon
Jumlah perhitungan rumus Euclidian : 2016

#####

Hasil dari run dengan algoritma DIVIDE AND CONQUER :

Pasangan titik terdekat      : [81.702, 0.868, -56.6] , [83.975, -16.828, -58.878]
Jarak pasangan titik terdekat : 17.986223311190148 satuan
Waktu eksekusi program       : 1.00048828125 milisekon
Jumlah perhitungan rumus Euclidean : 231

#####
```



n = 128, dimensi = 3

```
Penyelesaian masalah closest pair pada 3 Dimensi
Masukkan jumlah titik (n): 128

#####

Hasil dari run dengan algoritma BRUTE FORCE :

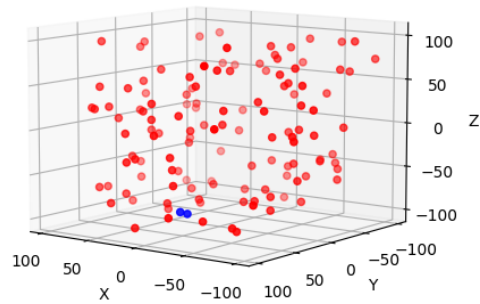
Pasangan titik terdekat      : [-24.14, 74.725, -80.744] , [-21.055, 80.09, -78.132]
Jarak pasangan titik terdekat : 6.71736510843352 satuan
Waktu eksekusi               : 4.00048828125 milisekon
Jumlah perhitungan rumus Euclidian : 8128

#####

Hasil dari run dengan algoritma DIVIDE AND CONQUER :

Pasangan titik terdekat      : [-24.14, 74.725, -80.744] , [-21.055, 80.09, -78.132]
Jarak pasangan titik terdekat : 6.71736510843352 satuan
Waktu eksekusi program       : 1.007080078125 milisekon
Jumlah perhitungan rumus Euclidean : 367

#####
```

n = 1000, dimensi = 3

Penyelesaian masalah closest pair pada 3 Dimensi

Masukkan jumlah titik (n): 1000

#####

Hasil dari run dengan algoritma BRUTE FORCE :

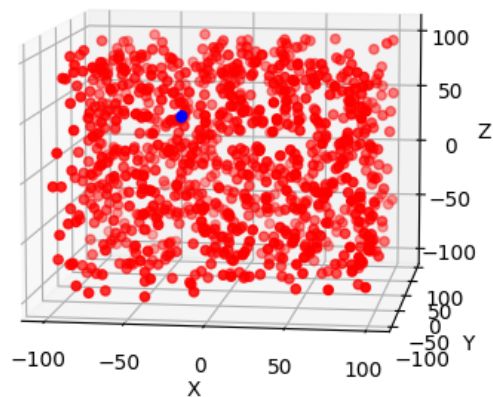
Pasangan titik terdekat : [-16.185, -96.147, 56.536] , [-16.576, -96.458, 55.341]
 Jarak pasangan titik terdekat : 1.2952324115771647 satuan
 Waktu eksekusi : 220.53369140625 milisekon
 Jumlah perhitungan rumus Euclidian : 499500

#####

Hasil dari run dengan algoritma DIVIDE AND CONQUER :

Pasangan titik terdekat : [-16.576, -96.458, 55.341] , [-16.185, -96.147, 56.536]
 Jarak pasangan titik terdekat : 1.2952324115771647 satuan
 Waktu eksekusi program : 43.0048828125 milisekon
 Jumlah perhitungan rumus Euclidean : 4188

#####



<p>n = 100, dimensi = 4</p>	<pre> Penyelesaian masalah closest pair pada N Dimensi Masukkan jumlah titik (n): 100 Masukkan dimensi: 4 ##### Hasil dari run dengan algoritma BRUTE FORCE : Pasangan titik terdekat : [98.845, -31.012, -68.12, -37.609] , [96.068, -23.981, -72.738, -25.074] Jarak pasangan titik terdekat : 15.349229264037982 satuan Waktu eksekusi : 5.0 milisekon Jumlah perhitungan rumus Euclidian : 4950 ##### Hasil dari run dengan algoritma DIVIDE AND CONQUER : Pasangan titik terdekat : [98.845, -31.012, -68.12, -37.609] , [96.068, -23.981, -72.738, -25.074] Jarak pasangan titik terdekat : 15.349229264037982 satuan Waktu eksekusi program : 0.998779296875 milisekon Jumlah perhitungan rumus Euclidean : 640 ##### </pre>
<p>n = 100, dimensi = 5</p>	<pre> Penyelesaian masalah closest pair pada N Dimensi Masukkan jumlah titik (n): 100 Masukkan dimensi: 5 ##### Hasil dari run dengan algoritma BRUTE FORCE : Pasangan titik terdekat : [76.351, 46.117, 41.768, 18.149, 32.401] , [68.588, 53.571, 40.477, 27.002, 41.138] Jarak pasangan titik terdekat : 16.498598243487233 satuan Waktu eksekusi : 5.512451171875 milisekon Jumlah perhitungan rumus Euclidian : 4950 ##### Hasil dari run dengan algoritma DIVIDE AND CONQUER : Pasangan titik terdekat : [76.351, 46.117, 41.768, 18.149, 32.401] , [68.588, 53.571, 40.477, 27.002, 41.138] Jarak pasangan titik terdekat : 16.498598243487233 satuan Waktu eksekusi program : 2.000244140625 milisekon Jumlah perhitungan rumus Euclidean : 975 ##### </pre>
<p>n = 100, dimensi = 6</p>	<pre> Penyelesaian masalah closest pair pada N Dimensi Masukkan jumlah titik (n): 100 Masukkan dimensi: 6 ##### Hasil dari run dengan algoritma BRUTE FORCE : Pasangan titik terdekat : [86.272, 51.835, 33.499, 22.201, -92.706, 5.004] , [96.686, 62.605, 18.929, 18.866, -74.968, 6.29] Jarak pasangan titik terdekat : 27.643115978485493 satuan Waktu eksekusi : 6.0 milisekon Jumlah perhitungan rumus Euclidian : 4950 ##### Hasil dari run dengan algoritma DIVIDE AND CONQUER : Pasangan titik terdekat : [86.272, 51.835, 33.499, 22.201, -92.706, 5.004] , [96.686, 62.605, 18.929, 18.866, -74.968, 6.29] Jarak pasangan titik terdekat : 27.643115978485493 satuan Waktu eksekusi program : 4.002197265625 milisekon Jumlah perhitungan rumus Euclidean : 1787 ##### </pre>

LAMPIRAN

Link Repository Github :

https://github.com/HobertJ/Tucil2_13521079

Berikut adalah Tabel Keberhasilan Program dalam memenuhi spesifikasi Tugas Kecil 2 Strategi Algoritma ,

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