

LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA

Penerapan Algoritma *Divide And Conquer* Untuk Mencari Titik Terdekat

Dosen Pengampu : Dr. Ir. Rinaldi, M.T



Disusun Oleh :

13521077 Husnia Munzayana

13521115 Shelma Salsabila

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022/2023

DAFTAR ISI

DAFTAR ISI	1
DAFTAR GAMBAR	2
DAFTAR TABEL	3
BAB I DESKRIPSI PERMASALAHAN	4
BAB II TEORI SINGKAT	5
2.1 Algoritma Divide and Conquer	5
2.2 Algoritma BruteForce	6
2.3 Rekursif.....	6
BAB III IMPLEMENTASI ALGORITMA.....	7
3.1 Implementasi Algoritma Divide And Conquer Untuk Mencari Titik Terdekat	7
3.2 Algoritma Brute Force	10
3.3 Algoritma Sorting dengan Quick Short	11
3.4 Algoritma Lain.....	12
3.5 Source Program.....	12
3.6 Link Pengerjaan	22
3.7 Check List Pengerjaan.....	22
BAB IV PERCOBAAN PROGRAM	23
BAB V KESIMPULAN DAN SARAN.....	31
5.1 Kesimpulan.....	31
5.2 Saran.....	31
DAFTAR PUSTAKA.....	32

DAFTAR GAMBAR

Gambar 1 Proses Divide and Conquer Secara Umum.....	8
Gambar 2 Proses Divide and Conquer Mencari Jarak Titik Terdekat	10
Gambar 3 Proses BruteForce Mencari Jarak Titik Terdekat	11
Gambar 4 Isi file tools.py.....	13
Gambar 5 Isi File Calculate.py	13
Gambar 6 Isi File CalculateDc.py	16
Gambar 7 Isi File plot.py	17
Gambar 8 Isi File quickSortSbX.py	18
Gambar 9 Isi File main.py	21

DAFTAR TABEL

Tabel 1 Check List Pengerjaan	22
Tabel 2 Percobaan-Percobaan Pada Program.....	30

BAB I

DESKRIPSI PERMASALAHAN

Mencari sepasang titik terdekat dengan Algoritma Divide and Conquer sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Pada Tugil 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 sepasang titik yang jaraknya terdekat satu sama lain dengan menerapkan algoritma divide and conquer untuk penyelesaiannya, dan perbandingannya dengan Algoritma Brute Force.

Adapun masukan program yang diinginkan adalah :

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

Adapun keluaran program yang diinginkan

- sepasang titik yang jaraknya terdekat dan nilai jaraknya
- banyaknya operasi perhitungan rumus Euclidian
- waktu riil dalam detik (spesifikasikan komputer yang digunakan)

Adapun Bonus yang dapat dikerjakan :

- 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 $x = (x_1, x_2, \dots, x_n)$

* Dikutip dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tucil2-Stima-2023.pdf>

BAB II

TEORI SINGKAT

2.1 Algoritma Divide and Conquer

Algoritma Divide and Conquer adalah algoritma pemecahan masalah dengan cara membagi masalah kedalam bagian-bagian kecil, kemudian menyelesaikan masalah tersebut dari bagian yang paling rendah / bawah. Hal ini dilakukan secara rekursif hingga mendapatkan solusi global sebagai penyelesaian masalah secara utuh. Tahapan penyelesaian dengan algoritma *divide and conquer* tersebut dapat dikelompokkan menjadi tiga tahapan, antara lain :

1. Divide

Pada tahapan ini, dilakukan pembagian persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula dengan ukuran yang lebih kecil (idealnya berukuran hampir sama). Pembagian dilakukan secara rekursif hingga mendapatkan upa-persoalan dengan skala se-minimum mungkin.

2. Conquer (solve)

Setelah persoalan global dipecah menjadi beberapa upa-persoalan, dilakukan pencarian solusi pada masing-masing upa-persoalan. Pencarian penyelesaian ini dilakukan pada upa-persoalan yang dianggap sudah memiliki skala paling kecil sehingga memudahkan dalam pencarian solusi.

3. Combine

Pada tahapan combine, dilakukan penggabungan solusi setiap upa-persoalan yang telah ditemukan. Tahapan ini dilakukan secara rekursif hingga dapat membentuk solusi global dari persoalan secara utuh.

Adapun algoritma ini bisa digunakan untuk menyelesaikan beberapa permasalahan, salah satunya adalah mencari sepasang titik terdekat (*closest pair problem*). Algoritma ini lebih mangkus dibandingkan algoritma greedy maupun brute force. Salah satu contohnya menyelesaikan permasalahan mencari jarak dua

titik terdekat dengan kompleksitasnya $O(n \log n)$. Hal ini tentu lebih mangkus daripada strategi brute force yang kompleksitasnya $O(n^2)$, sehigga pencarian solusi persoalan dapat dilakukan lebih cepat dengan algoritma divide and conquer. Penerapan algoritma ini menggunakan rekursif untuk penjelasan rekursif lebih jauh dibahas di 2.3

2.2 Algoritma BruteForce

Algoritma Brute Force adalah strategi pendekatan yang lempang (straightforward) untuk memecahkan suatu persoalan dengan sangat sederhana, langsung, dan dengan cara yang jelas. Contoh pengimplementasian algoritma ini adalah pada persoalan pencarian elemen terbesar/terkecil, sequential search, algoritma pengurutan (selection sort dan bubble sort), dan analisis pencocokan string. Penyelesaian persoalan dengan algoritma Brute Force umumnya menghasilkan solusi yang kurang efisien dan relatif lambat, walaupun terdapat beberapa permasalahan yang hanya dapat di selesaikan secara Brute Force. Hal ini disebabkan karena dalam algoritma Brute Force dipertimbangkan untuk menggunakan cara yang sederhana serta implementasi yang relatif cukup mudah dipahami, dibandingkan harus memikirkan cara yang rumit yang lebih efisien. Akan tetapi, *Algoritma Brute Force* umumnya dapat diterapkan pada hampir setiap permasalahan dan lebih mudah dimengerti karena algoritma pemecahan masalah yang digunakan relatif lebih sederhana.

2.3 Rekursif

Fungsi rekursif dalam pemrograman merupakan fungsi yang memanggil dirinya sendiri. Fungsi rekursif sering dibayangkan seperti perulangan. Karena tingkah lakunya yang mengulang-ulang setiap pemanggilan dirinya. Ada beberapa kelebihan dan kekurangan dalam rekursif ini . Kelebihannya yakni program lebih singkat, lebih efisien dan cepat dibandingkan proses secara iteratif. Adapun kekurangannya adalah memakan memori lebih besar dan seringkali menyebabkan program hang. Pada algoritma divide and conquer proses rekursif diterapkan pada tahap conquer(solve).

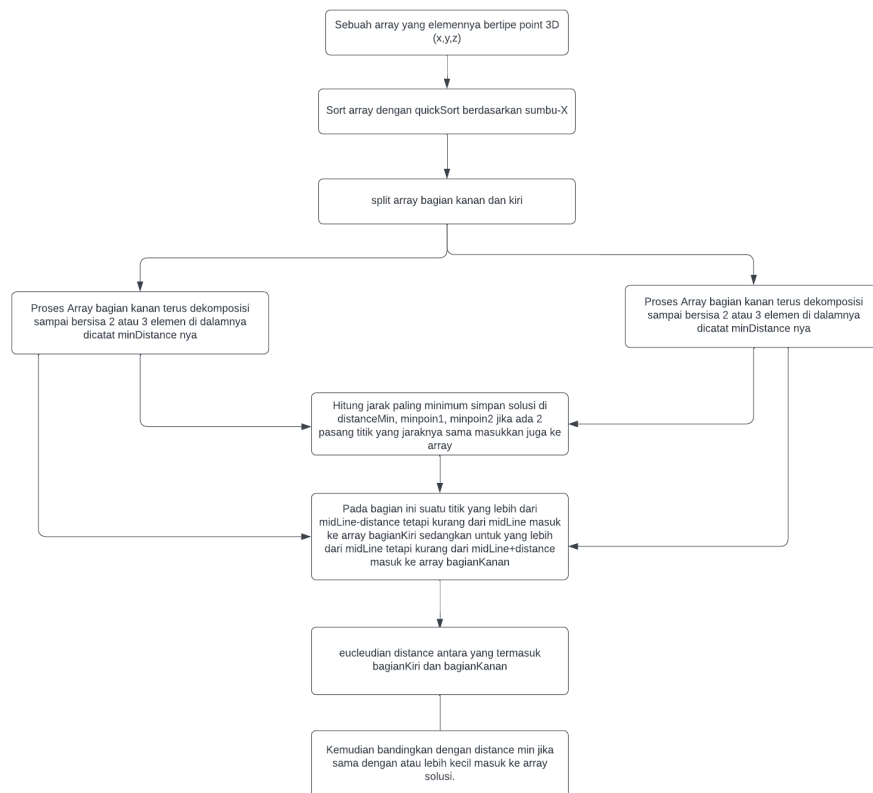
BAB III

IMPLEMENTASI ALGORITMA

3.1 Implementasi Algoritma Divide And Conquer Untuk Mencari Titik Terdekat

Persoalan mencari sepasang titik terdekat dengan algoritma *divide and conquer* dapat dilakukan dengan langkah-langkah berikut.

1. Program meminta input dari user mengenai dimensi vektor dalam variabel *d* serta banyaknya titik yang akan dibangkitkan disimpan dalam sebuah nilai *n*. Input nilai dimensi dan banyak titik akan divalidasi dengan aturan nilai dimensi harus lebih dari 0 dan banyak titik harus lebih dari 1.
2. Program akan membuat titik dalam dimensi *d* sebanyak *n* buah titik. Pembuatan titik ini dalam program dilakukan dengan fungsi `def randomPoint(n, d)`.
3. Titik – titik yang telah digenerate secara *random* tersebut disimpan dalam bentuk array. Array-array ini kemudian dibagi 2 dan disimpan dalam sebuah variabel lain. Yaitu array sebelah kiri *left* dan array sebelah kanan *right*. Kemudian terus dipisah sampai bersisa 3 atau 2. Kemudian dari kedua itu dicari yang jarak euclidean distance nya paling kecil kemudian disimpan ke *distanceMin*. Terus diproses sampai semuanya sudah dibandingkan. Kemudian diperiksa bagian tengah. Proses dari bagian tengah ini awalnya penulis menentukan garis semu yang posisinya ada pada *x* awal ditambah *x* akhir dibagi dua. Kemudian penulis menentukan titik sebelah kiri dari garis tengah dan wilayah titik sebelah kanan dari garis tengah disimpan dalam array kemudian dihitung jaraknya. Secara lebih jelas dapat digambarkan dalam bentuk flowchart sebagai berikut.



Gambar 1 Proses Divide and Conquer Secara Umum

Adapun secara pseudocode dapat dijelaskan sebagai berikut.

```

Algoritma Divide and Conquer Mencari Titik Terdekat
{inisiasi awal}
Points = [] {Sebuah array yang sudah diurutkan}
{Proses rekursif}
if(len(Points) = 2) then
    minDistance <- distance(points[0], points[1])
    minPoin1 <- Points[0]
    minPoin2 <- Points[1]
    return minPoin1, minPoin2, minDistance
else if(len(Points) = 3) then
    {compare distance}
    minDistance1 <- distance(points[0], points[1])
    minDistance2 <- distance(points[0], points[2])
    minDistance3 <- distance(points[1], points[2])
    {mencari nilai nilai paling minimum diantara ketiga nilai minDistance}
    minDistance <- min(minDistance1, minDistance2, minDistance3)
    {Kemudian nilai 2 point yang memiliki nilai distance yang sama dengan
    minDistance disimpan dalam array minPoin1 dan minPoin2}
  
```

```

    return minPoint1, minPoint2, minDistance

else {Kondisi len(Points) > 3}
    {Bagi 2 array}
    Left = [] //Bagian array hasil bagi 2 bagian kiri
    Right = [] // Bagian array hasil bagi 2 bagian kanan
    {dilakukan proses rekursif untuk bagian kiri}
    a,b,c = rekursif(left)
    d,e,f = rekursif(right)
    if(c > f)then
        distanceMin <- f
        minPoint1 = d
        minPoint2 = e
    else{kondisi f > c}
        distanceMin <- c
        minPoint1 = a
        minPoint2 = b
    {proses bagian tengah}
    lineMiddle <- (xpalingKecil+xpalingBesar)/2
    {Batasan nilai tengah}
    leftBoundary <- lineMiddle - distanceMin
    rightBoundary <- lineMiddle + distanceMin
    {Array yang menampung titik yang termasuk kedalam Batasan nilai
tengah}
    rightMiddle <- []
    leftMiddle <- []
    i transversal (0..len(Points))
        if(leftBoundary <= Left[i].x <= lineMiddle)then
            leftMiddle.append
        else if(lineMiddle <= Right[i].x <= rightBoundary)then
            rightMiddle.append
    i transversal[0..i]
        j transversal [0..j]
        {jika lebih kecil atau sama dengan di assign ke array solusi
minpoint1 dan minpoint2}
        g,h,i = rekursif(Left,Right)

    return minpoint1, minpoint2, minDistance

```

3.2 Algoritma Brute Force

Untuk menjadi pembanding algoritma divide and conquer digunakan algoritma brute force untuk menyelesaikan persoalan mencari sepasang titik terdekat. Algoritma brute force adalah algoritma yang straightforward pada dasarnya kebenaran dari algoritma ini dapat dipastikan. Sehingga, program yang dibuat dengan divide and conquer diharapkan sama kebenarannya dengan algoritma ini. Hanya saja waktu eksekusi program brute force lebih lambat dibandingkan dengan divide and conquer. Untuk algoritma brute force ini sama halnya dengan algoritma divide and conquer akan dihasilkan sepasang titik terdekat dan waktu eksekusi program. Nantinya hasil ini akan dianalisis kesesuaiannya dengan teori yang ada. Adapun Langkah mencari titik terdekat dengan algoritma brute force adalah sebagai berikut.

1. Program meminta input dari user mengenai banyaknya titik yang akan dibangkitkan disimpan dalam sebuah nilai n . Kemudian program meminta input dari user mengenai bidang yang ingin dibentuk seperti 3D, 4D atau yang lainnya yang selanjutnya disimpan dalam variabel m .
2. Program akan membuat titik dengan derajat m sebanyak n . Pembuatan titik ini dalam program dilakukan dengan fungsi `def randomPoint(n,m)`. Input m dan n akan divalidasi, m harus lebih dari 0. Dan n harus lebih dari 1.
3. Titik pertama atau elemen indeks ke 0 pada array tersebut di assign ke nilai `minPoint1`, kemudian elemen indeks ke 1 pada array tersebut di assign ke nilai `minPoint2` kemudian dicari distance nya. Jarak disimpan di sebuah variabel `minDistance`.

Program akan looping sampai semua titik dalam array dicari jaraknya, dan dibandingkan jarak yang paling kecil akan disimpan di `minDistance`. Adapun point-point yang memiliki jarak sebesar `minDistance` akan disimpan di variabel `minPoint1` dan `minPoint2`. Untuk algoritma ini dapat dijelaskan lebih singkat sebagai berikut.

Algoritma Mencari Jarak Terdekat Dengan BruteForce

```
{inisiasi Awal}
```

```
Points = [] {Sebuah array berisi sejumlah n titik dengan derajat m
```

```

minPoint1 = []
minPoint2 = []
minDistance <- distance(Points[0],Points[1])
i transversal(0..len(Points))
  j transversal(i+1..len(Points))
    dis <- distance(Points[i],Points[j])
    if(minDistance >= dis)then
      minDistance <- dis
      {Point[i] di assign ke minPoint1,
       Point[j] di assign ke minPoint2
    }
return minPoint1, minPoint2, minDistance

```

Gambar 3 Proses BruteForce Mencari Jarak Titik Terdekat

3.3 Algoritma Sorting dengan Quick Short

Proses sorting dengan algoritma sort dilakukan dengan cara divide and conquer

- Proses Divide

Proses divide terjadi ketika kita menentukan pivot lalu elemen array yang lebih kecil dari pivot disimpan dibagian kiri dan elemen array yang lebih besar dari pivot disimpan di bagian kanan.

- Proses Conquer

Yaitu proses rekursif dengan melakukan quick short Kembali pada array sebelah kanan dan array sebelah kiri hasil divide. Hal ini dilakukan terus menerus sampai terurut.

- Proses Combine

Array hasil pengurutan hasil divide and conquer kemudian digabung menjadi solusi array yang terurut dari besar ke kecil.

3.4 Algoritma Lain

Pada dasarnya algoritma lain ini tidak menggunakan strategi-strategi algoritma yang telah dipelajari. Hanya berisi penjelasan mengenai Langkah-langkah tambahan yang dilakukan untuk menyempurnakan tugas yang dibuat oleh penulis.

1. Program menggambar titik ke dalam bidang 3D

Pada program yang penulis buat penggambaran secara visual hanya bisa

digambarkan untuk program yang memiliki dimensi 3. Untuk selain itu penulis tidak menggambarannya dalam sebuah visualisasi. Adapun untuk penggambaran ini penulis menggunakan library-library yang ada di python yaitu matplotlib.pyplot.

2. Menghitung waktu eksekusi program

Perhitungan waktu eksekusi program dilakukan dengan mengimport library time pada python. Ada 2 waktu perhitungan yaitu untuk algoritma brute force dan divide and conquer.

3.5 Source Program

Untuk menyelesaikan persoalan penulis mendekomposisi penyelesaian ke dalam beberapa file. Berikut file-file itu serta penjelasannya.

1. File tools.py

File ini berisi alat-alat untuk input dan output program.

```
import math
import random

# Fungsi randomize point
def randomPoint(n, d):
    point = []
    for i in range(n):
        temp = []
        for i in range(d):
            temp.append(round(random.uniform(-100, 100), 2))
        point.append(temp)
    return point

# Function menghitung euclidean distance
def euclidDistance(point1, point2, countEuclid):
    countEuclid += 1
    # print(countEuclid)
    dis = 0
    for i in range(len(point1)):
        temp = point2[i] - point1[i]
        dis += math.pow(temp, 2)

    return round(math.sqrt(dis), 10), countEuclid
```

```
# Formatting output point
def printPoint(point):
    print("(", end='')
    for i in range(len(point)):
        print(point[i], end='')
        if (i == len(point) - 1):
            print(")")
        else:
            print(", ", end='')
```

Gambar 4 Isi file tools.py

2. File calculateBruteForce.py

File ini berisi fungsi untuk menyelesaikan persoalan dengan brute force

```
import math
from tools import *
def findPairBF(listPoint, countEuclid):
    minDistance, countEuclid = euclidDistance(
        listPoint[0], listPoint[1], countEuclid)
    countEuclid = 0
    minPoint1 = []
    minPoint2 = []
    for i in range(len(listPoint)):
        for j in range(i+1, len(listPoint)):
            dis, countEuclid = euclidDistance(
                listPoint[i], listPoint[j], countEuclid)
            if (minDistance == dis):
                minPoint1.append(listPoint[i])
                minPoint2.append(listPoint[j])
            elif (minDistance > dis):
                minDistance = dis
                minPoint1 = [listPoint[i]]
                minPoint2 = [listPoint[j]]
    return minDistance, minPoint1, minPoint2, countEuclid
```

Gambar 5 Isi File Calculate.py

3. File calculateDC.py

File ini berisi fungsi untuk menyelesaikan persoalan dengan divide and conquer

```
import math
import numpy
from tools import *
'''
Algoritma Divide Conquer
Divide : Bagi himpunan titik menjadi 2 bagian seimbang
         (jumlah titik tiap bagian sama)
Conquer : Solve jika jumlah titik suatu bagian 2 atau 3, cari
          distance minimum
Combine : Tentukan nilai jarak antar point minimum, terdapat 3
          kasus:
          (a) Pasangan titik terdekat terdapat pada bagian
              kanan (rightPoint)
          (b) Pasangan titik terdekat terdapat pada bagian
              kiri (leftPoint)
          (c) Pasangan titik terdekat terdapat pada pasangan
              titik yang dipisahkan garis maya pembagi (midPoint)
'''
# Fungsi menghitung jarak terdekat pasangan titik yang
# dipisahkan garis pembagi
def middleCompare(distanceMin, leftPoints, rightPoints,
                  point1, point2, midLine, countEuclid):
    inLeftMiddle = []
    inRightMiddle = []
    #rightboundary, cnt =
    euclidDistance(points[len(points)//2],
                    points[(len(points)+1)//2], countEuclid)
    rightboundary = midLine+distanceMin
    #leftboundary, cnt2 =
    euclidDistance(points[len(points)//2],
                    points[(len(points)-1)//2], countEuclid)
    leftboundary = midLine-distanceMin
    # print("boundaries", rightboundary, leftboundary)
    for point in leftPoints:
        if leftboundary <= point[0] <= midLine:
            inLeftMiddle.append(point)
    for point in rightPoints:
        if midLine <= point[0] <= rightboundary:
            inRightMiddle.append(point)
    for i in range(len(inLeftMiddle)):
        for j in range(len(inRightMiddle)):
```

```

        distanceNow, countEuclid = euclidDistance(
            inLeftMiddle[i], inRightMiddle[j],
countEuclid)
        if (distanceMin == distanceNow):
            point1.append(inLeftMiddle[i])
            point2.append(inRightMiddle[j])
        elif distanceMin > distanceNow:
            point1 = [inLeftMiddle[i]]
            point2 = [inRightMiddle[j]]
            distanceMin = distanceNow
    return distanceMin, point1, point2, countEuclid

# Menemukan pasangan titik dengan jarak terdekat
def findPairDC(point, countEuclid):
    # Proses Conquer / solve
    if (len(point) == 2):
        minDistance, countEuclid = euclidDistance(
            point[0], point[1], countEuclid)
        minPoint1 = [point[0]]
        minPoint2 = [point[1]]
    elif (len(point) == 3):
        # Compare P0 dan P1
        minDistance, countEuclid = euclidDistance(
            point[0], point[1], countEuclid)
        minPoint1 = [point[0]]
        minPoint2 = [point[1]]

        # Compare P0 dan P2
        tempDistance, countEuclid = euclidDistance(
            point[0], point[2], countEuclid)
        if (minDistance == tempDistance):
            minPoint1.append(point[0])
            minPoint2.append(point[2])
        elif (minDistance > tempDistance):
            minDistance = tempDistance
            minPoint1 = [point[0]]
            minPoint2 = [point[2]]

        # Compare P1 dan P2
        tempDistance, countEuclid = euclidDistance(

```



```

        point[1], point[2], countEuclid)
    if (minDistance == tempDistance):
        minPoint1.append(point[1])
        minPoint2.append(point[2])
    elif (minDistance > tempDistance):
        minDistance = tempDistance
        minPoint1 = [point[1]]
        minPoint2 = [point[2]]
else:
    # Proses Divide
    leftPoint = point[: (len(point)//2)]
    rightPoint = point[((len(point)//2)):]
    midPoint = (leftPoint[len(leftPoint)-1][0] +
rightPoint[0][0])/2
    leftDistance, leftPoint1, leftPoint2, countEuclid =
findPairDC(
        leftPoint, countEuclid)
    rightDistance, rightPoint1, rightPoint2, countEuclid =
findPairDC(
        rightPoint, countEuclid)
    # Proses Combine
    # print("\n-- Combine --")
    if (leftDistance == rightDistance):
        minDistance = leftDistance
        minPoint1 = leftPoint1 + rightPoint2
        minPoint2 = leftPoint2 + rightPoint2
    elif (leftDistance < rightDistance):
        minDistance = leftDistance
        minPoint1 = leftPoint1
        minPoint2 = leftPoint2
    else:
        minDistance = rightDistance
        minPoint1 = rightPoint1
        minPoint2 = rightPoint2
    minDistance, minPoint1, minPoint2, countEuclid =
middleCompare(
        minDistance, leftPoint, rightPoint, minPoint1,
minPoint2, midPoint, countEuclid)

return minDistance, minPoint1, minPoint2, countEuclid

```

Gambar 6 Isi File CalculateDc.py

4. File plot.py

File ini berisi fungsi untuk memvisualisasikan titik-titik pada array input pada sebuah bidang. Visualisasi hanya bisa dilakukan pada bidang 3D

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

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

def drawPlotResult(minPoint1,minPoint2):
    setColor = ['red', 'blue', 'orange',
                'darkorchid', 'green', 'black', 'yellow']
    for i in range(len(minPoint1)):
        x, y, z = minPoint1[i]
        ax.plot([x], [y], [z], marker='o',markersize=7,
                color=setColor[i % 6])
        x,y,z = minPoint2[i]
        ax.plot([x], [y], [z], marker='o',markersize=7,
                color=setColor[i % 6])

def drawPlot(points,minPoint1,minPoint2):
    for point in points:
        if(point not in minPoint1) and (point not in
        minPoint2):
            x, y, z = point
            ax.plot([x], [y], [z], marker='o',markersize=5,
                    color='darkgrey')

    # Set the labels for the x, y, and z axes
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    # Show the plot
    plt.show()
```

Gambar 7 Isi File plot.py

5. File quickSortSbX.py

Di dalam file ini berisi kode untuk sorting titik berdasarkan sumbu x dengan quick sort

```
#Fungsi sort dengan Quick Sort Menggunakan Divide And Conquer
#Diurutkan berdasarkan sb-X dari kecil ke besar
#Pivot yang digunakan adalah elemen terakhir dari array
def partitionSort(point,idxInit,idxLast):
    #Pivot elemen terakhir array
    pivot = point[idxLast][0]
    #Bagian ini mempartisi dengan cara nilai-nilai yang lebih
        kecil dari pivot ditaruh
    #dibagian paling kiri dan nilai yang lebih besar dari pivot
        ditaruh di bagian kanan pivot
    #nilai pertama yang ditemui lebih kecil dari pivot akan
        ditaruh paling ujung kiri dan seterusnya
    #Fungsi mereturn nilai dimana partisi telah berhenti
        dilakukan
    i = idxInit-1
    for j in range(idxInit,idxLast):
        if point[j][0] <= pivot:
            i = i+1
            point[i],point[j] = point[j],point[i] #Bagian swap
    point[i+1],point[idxLast] = point[idxLast],point[i+1]
    return ( i+1 )

#Main quick sort
def quickSortSbX(point,idxInit,idxLast):
    if idxInit < idxLast:
        partisi = partitionSort(point, idxInit, idxLast)
        quickSortSbX(point, idxInit, partisi-1) #rekursif
        bagian kiri array
        quickSortSbX(point, partisi+1, idxLast) #rekursif
        bagian kanan array
    return point
```

Gambar 8 Isi File quickSortSbX.py

6. File main.py

File ini berisi main program persoalan, semua output akan bisa dihasilkan ketika kita menjalankan program pada bagian main

```

print()
proc = '\033[01m\033[33mProcessing ...\033[0m'
for i in proc:
    print(i, end='')
    time.sleep(0.3)
print("\n")

point = randomPoint(n, d)      # Menghasilkan array of point
# point.append([0, 0, 0])

point = quickSortSbX(point, 0, len(point)-1)

# Pencarian Pasangan Titik Terdekat dengan Algoritma
  BruteForce
startBF = time.time()
minDistance, minPoint1, minPoint2, countEuclid =
    findPairBF(point, 0)

# Menghitung Waktu Eksekusi
exeTimeBF = time.time() - startBF

# Output Program
print("\033[1m\033[95m==== ALGORITMA BRUTE FORCE ==== \033[0m")
print("\033[95mTerdapat\033[01m", len(minPoint1),
      "pasang \033[0m\033[95mtitik terdekat !")
print("Jarak pasangan titik terdekat :", minDistance, "satuan
      jarak\033[0m")
for i in range(len(minPoint1)):
    print("\nPasangan Titik Terdekat ke-", (i+1), " :")
    print("  Titik 1 : ", end='')
    printPoint(minPoint1[i])
    print("  Titik 2 : ", end='')
    printPoint(minPoint2[i])
print("\n\033[95mBanyak Perhitungan Jarak Euclidean Distance
      :", countEuclid)
print("Execution Time : %s seconds" % exeTimeBF, "\033[0m")

# Pencarian Pasangan Titik Terdekat dengan Algoritma Divide
  and Conquer
startDC = time.time()
minDistance, minPoint1, minPoint2, countEuclid =

```

```

        findPairDC(point, 0)

# Menghitung Waktu Eksekusi
exeTimeDC = time.time() - startDC

# Output Program
print(
    "\n\033[1m\033[92m==== ALGORITMA DIVIDE AND CONQUER
    ====\033[0m")
print("\033[92mTerdapat\033[01m", len(minPoint1),
      "pasang \033[0m\033[92mtitik terdekat !")
print("Jarak pasangan titik terdekat :", minDistance, "satuan
      jarak\033[0m")
for i in range(len(minPoint1)):
    print("\nPasangan Titik Terdekat ke-", (i+1), " :")
    print("  Titik 1 : ", end='')
    printPoint(minPoint1[i])
    print("  Titik 2 : ", end='')
    printPoint(minPoint2[i])
print("\n\033[32mBanyak Perhitungan Jarak Euclidean Distance
      :", countEuclid)
print("Execution Time : %s seconds" % exeTimeDC, "\033[0m")

# Visualisasi
if(d == 3):
    print("\n\033[01m\033[33mShow 3D Visualization
    ...\033[0m")
    drawPlotResult(minPoint1, minPoint2)
    drawPlot(point, minPoint1, minPoint2)

print()

```

Gambar 9 Isi File main.py

3.6 Link Pengerjaan

https://github.com/munzayanahusn/Tucil2_13521077_13521115.git

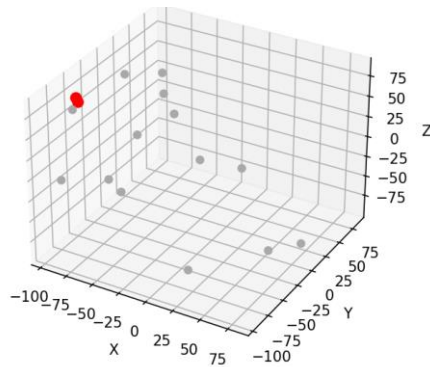
3.7 Check List Pengerjaan

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

Tabel 1 Check List Pengerjaan

BAB IV

PERCOBAAN PROGRAM

Percobaan	Kondisi	Output hasil									
1	Jumlah titik : 16 Jumlah derajat : 3	<div style="background-color: #2e3436; color: #eeeeec; padding: 10px; font-family: monospace;"> <h3 style="text-align: center; margin: 0;">CLOSEST PAIR</h3> <p>Masukkan dimensi vektor : 3 Masukkan jumlah titik : 16</p> <p style="color: #f0f0f0;">Processing ...</p> <p>==== ALGORITMA BRUTE FORCE ====</p> <p>Terdapat 1 pasang titik terdekat ! Jarak pasangan titik terdekat : 6.2274392811 satuan jarak</p> <p>Pasangan Titik Terdekat ke- 1 : Titik 1 : (-81.19, -75.63, 91.21) Titik 2 : (-80.48, -73.0, 85.61)</p> <p>Banyak Perhitungan Jarak Euclidean Distance : 120 Execution Time : 0.0009205341339111328 seconds</p> <p>==== ALGORITMA DIVIDE AND CONQUER ====</p> <p>Terdapat 1 pasang titik terdekat ! Jarak pasangan titik terdekat : 6.2274392811 satuan jarak</p> <p>Pasangan Titik Terdekat ke- 1 : Titik 1 : (-81.19, -75.63, 91.21) Titik 2 : (-80.48, -73.0, 85.61)</p> <p>Banyak Perhitungan Jarak Euclidean Distance : 32 Execution Time : 0.0 seconds</p> <p style="color: #f0f0f0;">Show 3D Visualization ...</p> </div> <div style="text-align: center; margin-top: 20px;">  </div> <p style="margin-top: 20px;">Dari percobaan 1 ini diperoleh informasi sebagai berikut.</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th></th><th><i>Brute Force</i></th><th><i>Divide and Conquer</i></th></tr> </thead> <tbody> <tr> <td>Jumlah Eucleudian Distance</td><td>120</td><td>32</td></tr> <tr> <td>Waktu eksekusi program</td><td>0.0009205 second</td><td>0 second</td></tr> </tbody> </table>		<i>Brute Force</i>	<i>Divide and Conquer</i>	Jumlah Eucleudian Distance	120	32	Waktu eksekusi program	0.0009205 second	0 second
	<i>Brute Force</i>	<i>Divide and Conquer</i>									
Jumlah Eucleudian Distance	120	32									
Waktu eksekusi program	0.0009205 second	0 second									

2

Jumlah Titik = 64

Jumlah Derajat = 3

```

CLOSEST PAIR

Masukkan dimensi vektor : 3
Masukkan jumlah titik : 64

Processing ...

==== ALGORITMA BRUTE FORCE ====
Terdapat 1 pasang titik terdekat !
Jarak pasangan titik terdekat : 9.4212366492 satuan jarak

Pasangan Titik Terdekat ke- 1 :
Titik 1 : (84.92, -16.6, -72.13)
Titik 2 : (88.97, -8.96, -68.39)

Banyak Perhitungan Jarak Euclidean Distance : 2016
Execution Time : 0.0030145645141601562 seconds

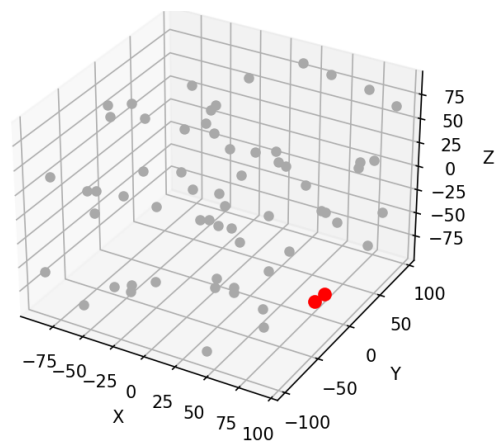
==== ALGORITMA DIVIDE AND CONQUER ====
Terdapat 1 pasang titik terdekat !
Jarak pasangan titik terdekat : 9.4212366492 satuan jarak

Pasangan Titik Terdekat ke- 1 :
Titik 1 : (84.92, -16.6, -72.13)
Titik 2 : (88.97, -8.96, -68.39)

Banyak Perhitungan Jarak Euclidean Distance : 383
Execution Time : 0.0 seconds

Show 3D Visualization ...

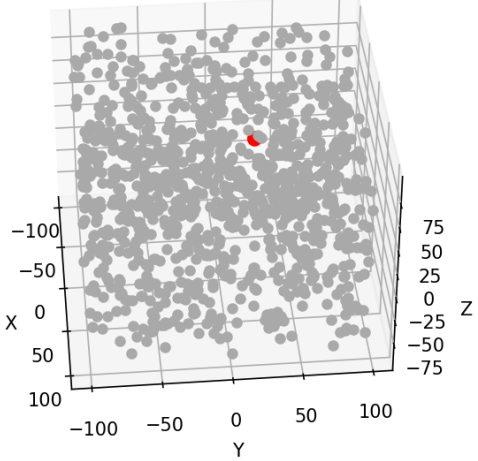
```

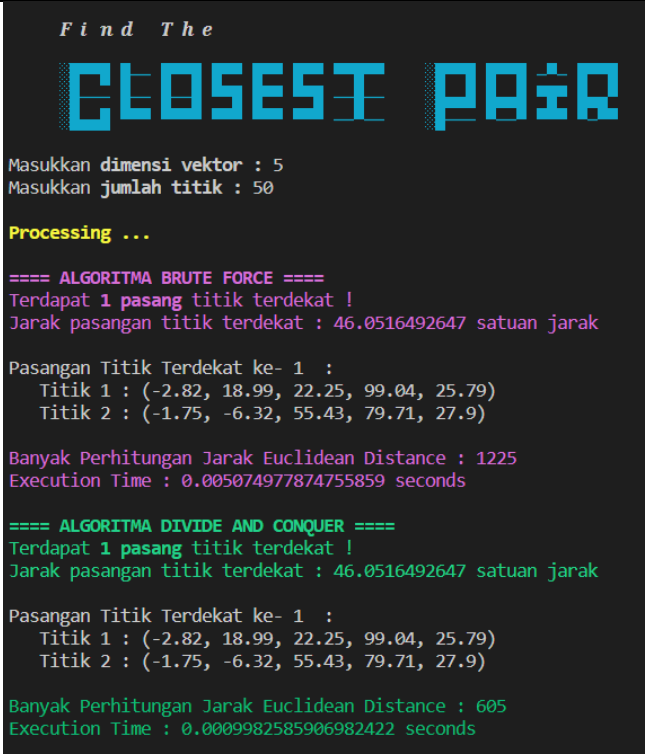


Dari percobaan 2 ini diperoleh informasi sebagai berikut.

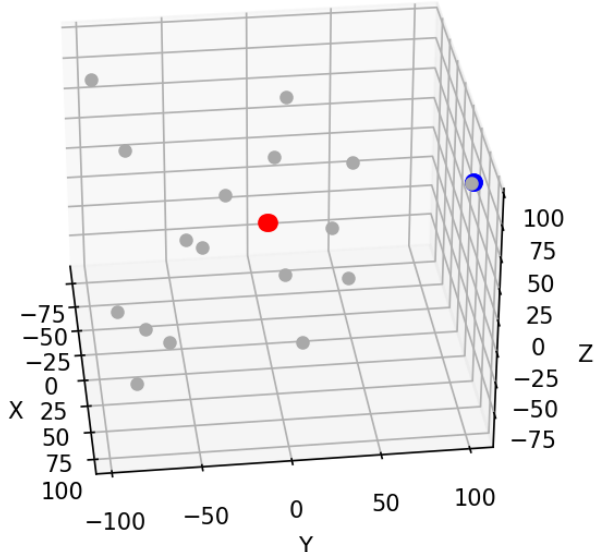
	<i>Brute Force</i>	<i>Divide and Conquer</i>
Jumlah Euclidean Distance	2016	383
Waktu eksekusi program	0.003015 second	0 second

3	<p>Jumlah Titik : 128</p> <p>Jumlah Derajat : 3</p>	<div data-bbox="798 262 1345 896"> <h2 style="text-align: center;">CLOSEST PAIR</h2> <p>Masukkan dimensi vektor : 3 Masukkan jumlah titik : 128</p> <p>Processing ...</p> <p>==== ALGORITMA BRUTE FORCE ====</p> <p>Terdapat 1 pasang titik terdekat ! Jarak pasangan titik terdekat : 6.5412766338 satuan jarak</p> <p>Pasangan Titik Terdekat ke- 1 : Titik 1 : (5.99, 90.65, 98.41) Titik 2 : (7.92, 84.4, 98.38)</p> <p>Banyak Perhitungan Jarak Euclidean Distance : 8128 Execution Time : 0.01611018180847168 seconds</p> <p>==== ALGORITMA DIVIDE AND CONQUER ====</p> <p>Terdapat 1 pasang titik terdekat ! Jarak pasangan titik terdekat : 6.5412766338 satuan jarak</p> <p>Pasangan Titik Terdekat ke- 1 : Titik 1 : (5.99, 90.65, 98.41) Titik 2 : (7.92, 84.4, 98.38)</p> <p>Banyak Perhitungan Jarak Euclidean Distance : 1013 Execution Time : 0.0019986629486083984 seconds</p> <p>Show 3D Visualization ...</p> </div> <div data-bbox="798 907 1377 1411"> </div> <table border="1"> <thead> <tr> <th></th><th><i>Brute Force</i></th><th><i>Divide and Conquer</i></th></tr> </thead> <tbody> <tr> <td>Jumlah Eucleudian Distance</td><td>8128</td><td>1013</td></tr> <tr> <td>Waktu eksekusi program</td><td>0.0161 second</td><td>0.00198 second</td></tr> </tbody> </table>		<i>Brute Force</i>	<i>Divide and Conquer</i>	Jumlah Eucleudian Distance	8128	1013	Waktu eksekusi program	0.0161 second	0.00198 second
	<i>Brute Force</i>	<i>Divide and Conquer</i>									
Jumlah Eucleudian Distance	8128	1013									
Waktu eksekusi program	0.0161 second	0.00198 second									

4	<p>Jumlah Titik : 1000</p> <p>Jumlah Derajat : 3</p>	<pre> ===== ALGORITMA BRUTE FORCE ===== Terdapat 1 pasang titik terdekat ! Jarak pasangan titik terdekat : 0.7123903424 satuan jarak Pasangan Titik Terdekat ke- 1 : Titik 1 : (45.82, 20.57, 90.75) Titik 2 : (46.37, 20.26, 90.42) Banyak Perhitungan Jarak Euclidean Distance : 499500 Execution Time : 1.040968894958496 seconds ===== ALGORITMA DIVIDE AND CONQUER ===== Terdapat 1 pasang titik terdekat ! Jarak pasangan titik terdekat : 0.7123903424 satuan jarak Pasangan Titik Terdekat ke- 1 : Titik 1 : (45.82, 20.57, 90.75) Titik 2 : (46.37, 20.26, 90.42) Banyak Perhitungan Jarak Euclidean Distance : 25009 Execution Time : 0.05398130416870117 seconds </pre>  <table border="1"> <thead> <tr> <th></th><th><i>Brute Force</i></th><th><i>Divide and Conquer</i></th></tr> </thead> <tbody> <tr> <td>Jumlah Euclidean Distance</td><td>499500</td><td>25009</td></tr> <tr> <td>Waktu eksekusi program</td><td>1.04096 second</td><td>0.053981 second</td></tr> </tbody> </table>		<i>Brute Force</i>	<i>Divide and Conquer</i>	Jumlah Euclidean Distance	499500	25009	Waktu eksekusi program	1.04096 second	0.053981 second
	<i>Brute Force</i>	<i>Divide and Conquer</i>									
Jumlah Euclidean Distance	499500	25009									
Waktu eksekusi program	1.04096 second	0.053981 second									

5	Jumlah Titik : 50 Dimensi : 5	 <pre> Find The CLOSEST PAIR Masukkan dimensi vektor : 5 Masukkan jumlah titik : 50 Processing ... ==== ALGORITMA BRUTE FORCE ==== Terdapat 1 pasang titik terdekat ! Jarak pasangan titik terdekat : 46.0516492647 satuan jarak Pasangan Titik Terdekat ke- 1 : Titik 1 : (-2.82, 18.99, 22.25, 99.04, 25.79) Titik 2 : (-1.75, -6.32, 55.43, 79.71, 27.9) Banyak Perhitungan Jarak Euclidean Distance : 1225 Execution Time : 0.005074977874755859 seconds ==== ALGORITMA DIVIDE AND CONQUER ==== Terdapat 1 pasang titik terdekat ! Jarak pasangan titik terdekat : 46.0516492647 satuan jarak Pasangan Titik Terdekat ke- 1 : Titik 1 : (-2.82, 18.99, 22.25, 99.04, 25.79) Titik 2 : (-1.75, -6.32, 55.43, 79.71, 27.9) Banyak Perhitungan Jarak Euclidean Distance : 605 Execution Time : 0.0009982585906982422 seconds </pre> <table border="1"> <thead> <tr> <th></th><th><i>Brute Force</i></th><th><i>Divide and Conquer</i></th></tr> </thead> <tbody> <tr> <td>Jumlah Eucleudian Distance</td><td>1225</td><td>605</td></tr> <tr> <td>Waktu eksekusi program</td><td>0.005 second</td><td>0.000998 second</td></tr> </tbody> </table>		<i>Brute Force</i>	<i>Divide and Conquer</i>	Jumlah Eucleudian Distance	1225	605	Waktu eksekusi program	0.005 second	0.000998 second
	<i>Brute Force</i>	<i>Divide and Conquer</i>									
Jumlah Eucleudian Distance	1225	605									
Waktu eksekusi program	0.005 second	0.000998 second									

6	<p>Percobaan dengan variasi dimensi</p> <p>Jumlah Titik : 500</p> <p>Dimensi : 10</p>	 <pre> Find The CLOSEST PAIR Masukkan dimensi vektor : 10 Masukkan jumlah titik : 500 Processing ... ==== ALGORITMA BRUTE FORCE ==== Terdapat 1 pasang titik terdekat ! Jarak pasangan titik terdekat : 53.8256751003 satuan jarak Pasangan Titik Terdekat ke- 1 : Titik 1 : (-90.21, -32.69, 23.13, 0.23, 80.29, 18.88, -10.55, -5.21, -23.31, 36.5) Titik 2 : (-54.35, -35.76, 27.24, 28.29, 96.83, 15.13, -9.72, -0.77, -1.39, 33.49) Banyak Perhitungan Jarak Euclidean Distance : 124750 Execution Time : 0.38696885108947754 seconds ==== ALGORITMA DIVIDE AND CONQUER ==== Terdapat 1 pasang titik terdekat ! Jarak pasangan titik terdekat : 53.8256751003 satuan jarak Pasangan Titik Terdekat ke- 1 : Titik 1 : (-90.21, -32.69, 23.13, 0.23, 80.29, 18.88, -10.55, -5.21, -23.31, 36.5) Titik 2 : (-54.35, -35.76, 27.24, 28.29, 96.83, 15.13, -9.72, -0.77, -1.39, 33.49) Banyak Perhitungan Jarak Euclidean Distance : 79673 Execution Time : 0.26234984397888184 seconds </pre> <table border="1" data-bbox="676 819 1477 1102"> <thead> <tr> <th></th><th><i>Brute Force</i></th><th><i>Divide and Conquer</i></th></tr> </thead> <tbody> <tr> <td>Jumlah Eucleudian Distance</td><td>124750</td><td>79673</td></tr> <tr> <td>Waktu eksekusi program</td><td>0.387 seconds</td><td>0.262 seconds</td></tr> </tbody> </table>		<i>Brute Force</i>	<i>Divide and Conquer</i>	Jumlah Eucleudian Distance	124750	79673	Waktu eksekusi program	0.387 seconds	0.262 seconds
	<i>Brute Force</i>	<i>Divide and Conquer</i>									
Jumlah Eucleudian Distance	124750	79673									
Waktu eksekusi program	0.387 seconds	0.262 seconds									
7	<p>Input jumlah titik dan dimensi yang tidak valid</p>	 <pre> Find The CLOSEST PAIR Masukkan dimensi vektor : 0 Input salah! Dimensi harus Bilangan Asli Masukkan dimensi vektor : -1 Input salah! Dimensi harus Bilangan Asli Masukkan dimensi vektor : 2 Masukkan jumlah titik : 1 Input salah! Banyak titik minimum 2 Masukkan jumlah titik : -1 Input salah! Banyak titik minimum 2 Masukkan jumlah titik : 2 Processing ... </pre>									

8.	Terdapat 2 pasangan array yang jaraknya sama	<div data-bbox="767 230 1385 976"> <pre> Find The CLOSEST PAIR Jarak pasangan titik terdekat : 1.7320508076 satuan jarak Pasangan Titik Terdekat ke- 1 : Titik 1 : (0, 0, 0) Titik 2 : (1, 1, 1) Pasangan Titik Terdekat ke- 2 : Titik 1 : (99, 98, 99) Titik 2 : (100, 99, 100) Banyak Perhitungan Jarak Euclidean Distance : 190 Execution Time : 0.0009164810180664062 seconds ==== ALGORITMA DIVIDE AND CONQUER ==== Terdapat 2 pasang titik terdekat ! Jarak pasangan titik terdekat : 1.7320508076 satuan jarak Pasangan Titik Terdekat ke- 1 : Titik 1 : (0, 0, 0) Titik 2 : (1, 1, 1) Pasangan Titik Terdekat ke- 2 : Titik 1 : (100, 99, 100) Titik 2 : (100, 99, 100) Banyak Perhitungan Jarak Euclidean Distance : 30 Execution Time : 0.0 seconds Show 3D Visualization ... </pre> </div> <div data-bbox="767 976 1366 1525">  </div>
----	--	---

9	Terdapat 3 pasang array yang nilainya sama	<div data-bbox="778 197 1375 1075"> <pre> Find The CLOSEST PAIR Masukkan dimensi vektor : 3 Masukkan jumlah titik : 17 Processing ... ==== ALGORITMA BRUTE FORCE ==== Terdapat 3 pasang titik terdekat ! Pasangan Titik Terdekat ke- 2 : Titik 1 : (13, 16, 17) Titik 2 : (14, 17, 18) Pasangan Titik Terdekat ke- 3 : Titik 1 : (99, 98, 99) Titik 2 : (100, 99, 100) Banyak Perhitungan Jarak Euclidean Distance : 253 Execution Time : 0.0010945796966552734 seconds ==== ALGORITMA DIVIDE AND CONQUER ==== Terdapat 3 pasang titik terdekat ! Jarak pasangan titik terdekat : 1.7320508076 satuan jarak Pasangan Titik Terdekat ke- 1 : Titik 1 : (13, 16, 17) Titik 2 : (14, 17, 18) Pasangan Titik Terdekat ke- 2 : Titik 1 : (100, 99, 100) Titik 2 : (100, 99, 100) Pasangan Titik Terdekat ke- 3 : Titik 1 : (0, 0, 0) Titik 2 : (1, 1, 1) Banyak Perhitungan Jarak Euclidean Distance : 43 Execution Time : 0.0 seconds </pre> </div> <div data-bbox="676 1111 1485 1805"> </div>
---	--	---

Tabel 2 Percobaan-Percobaan Pada Program

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil percobaan yang dilakukan pada 3 buah kondisi dengan derajat yang sama yaitu n dengan jumlah titik yang berubah terus menaik. Kita bisa melihat bahwa dalam menyelesaikan *closest pair*, algoritma *divide and conquer* lebih mangkus dibanding dengan algoritma *brute force*. Hal ini dapat dilihat dari banyaknya perhitungan *euclidean distance* dengan algoritma *divide and conquer* jauh lebih dibandingkan dengan algoritma *brute force*. Hal ini cukup berpengaruh pada kompleksitas waktu dari program. Dapat kita bandingkan bahwa, secara umum eksekusi program *brute force* lebih lama dibandingkan dengan *divide and conquer*. Hal ini sesuai dengan teori yang telah dibahas sebelumnya bahwa kompleksitas algoritma untuk *brute force* dalam pencarian *closest pair* adalah $O(n^2)$, sedangkan kompleksitas *divide and conquer* adalah $O(n \log n)$. Pembuktian ini menunjukkan algoritma *divide and conquer* yang penulis terapkan dalam program sudah dapat berjalan dengan baik dan memberikan hasil yang cukup valid.

5.2 Saran

Penulis berharap untuk kedepannya dapat mengerjakan dengan lebih maksimal serta dapat membandingkan beberapa lebih banyak algoritma lain untuk menemukan solusi paling optimum untuk persoalan penentuak pasangan titik terdekat atau *closest pair*.

DAFTAR PUSTAKA

- Munir, Rinaldi (2022). Algoritma Divide And Conquer Bagian 2. Diakses pada 26 Februari 2023 pukul 13.00 dari 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)
- Munir, Rinaldi (2022). Algoritma Brute Force Bagian 1. Diakses pada 26 Februari 2023 pukul 15.00 dari sumber [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-\(2022\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf)