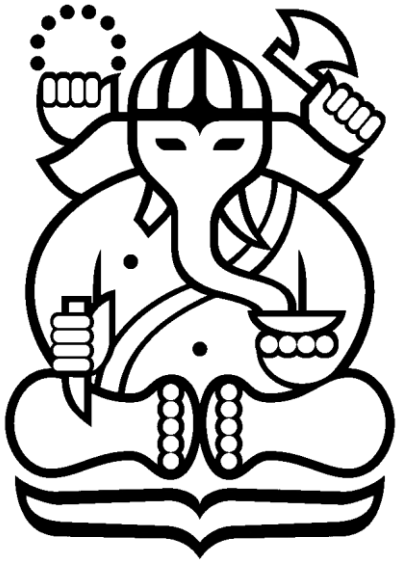
LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA

PENCARIAN PASANGAN TITIK TERDEKAT DALAM RUANG TIGA DIMENSI DENGAN ALGORITMA *DIVIDE AND CONQUER*

Dosen Pengajar: Dr. Nur Ulfa Maulidevi, S.T., M.Sc.



Disusun oleh:

Kevin John Wesley Hutabarat (13521042)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
FEBRUARI 2023

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1
DESKRIPSI TUGAS

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 P1 = (x_1, y_1, z_1) dan P2 = (x_2, y_2, z_2) dihitung dengan rumus Euclidean berikut:

$$d = \sqrt[3]{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

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 R^n, setiap vektor dinyatakan dalam bentuk x = (x_1, x_2,…, x_n)

BAB 2
LANDASAN TEORI

1. Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* adalah algoritma yang menyelesaikan suatu persoalan dengan cara membagi persoalan tersebut menjadi bagian yang lebih kecil agar lebih mudah diselesaikan. Algoritma ini mengadopsi strategi militer yang dikenal dengan istilah *divide ut imperes*. Secara garis besar, algoritma ini memiliki tiga tahapan, yaitu *divide*, *conquer*, dan *combine*.

Pada tahap *divide*, algoritma memecah persoalan menjadi persoalan-persoalan yang sama tetapi dalam cakupan yang lebih kecil. Masing-masing pecahan persoalan tersebut diselesaikan pada tahap *conquer* (*solve*). Setelah setiap upa-persoalan diselesaikan, selanjutnya solusi akan digabungkan pada tahap *combine*, sehingga membentuk solusi dari permasalahan semula.

Algoritma ini umumnya menyelesaikan persoalan dengan metode rekursif, dengan basisnya adalah persoalan yang tidak dapat dibagi lagi ke persoalan yang lebih kecil. Objek persoalan yang dibagi dapat berupa larik, matriks, eksponen, polinom, dll. Setiap *instance* yang dibagi harus memiliki karakteristik yang sama dengan persoalan semula. Kompleksitas dari algoritma ini adalah:

$$T(n) = \begin{cases} g(n), n \leq n_0 \\ T(n_1) + T(n_2) + \cdots + T(n_r) + f(n), n > n_0 \end{cases}$$

Dengan T(n) adalah kompleksitas waktu penyelesaian persoalan yang berukuran n, g(n) adalah kompleksitas waktu untuk menyelesaikan n berukuran kecil, dan f(n) adalah waktu untuk menggabungkan solusi persoalan.

2. Penerapan Algoritma *Divide and Conquer* untuk Mencari Pasangan Titik Terdekat

Algoritma *Divide and Conquer* dapat digunakan untuk menyelesaikan persoalan pasangan titik terdekat. Langkah-langkah untuk mencari pasangan titik terdekat adalah:

1. Bangkitkan titik secara acak, dengan parameter dimensi dan banyak titik
2. Urutkan titik dalam himpunan berdasarkan nilai absis secara menaik.
3. Jika banyak titik hanya dua, maka jarak dari kedua titik dihitung langsung dengan rumus jarak euclidean.
4. Jika banyak titik hanya tiga, maka jarak dari masing-masing titik akan dihitung dengan rumus jarak euclidean, kemudian diambil jarak terkecil beserta pasangan yang memiliki jarak tersebut.

5. Jika banyak titik lebih dari tiga, bagi himpunan titik ke dalam dua bagian, `arrayOfPoint1` dan `arrayOfPoint2`, dengan setiap bagian memiliki jumlah titik yang sama atau hanya berselisih satu titik.
6. Pada kedua bagian tersebut, algoritma *divide and conquer* diterapkan secara rekursif untuk mencari sepasang titik yang terdekat.
7. Bandingkan hasil dari kedua bagian dan ambil jarak yang terkecil, simpan dalam variabel `d`.
8. Tinjau kasus apabila jarak terdekat berada di antara satu titik di bagian pertama dan satu titik di bagian kedua. Untuk setiap titik di bagian pertama, akan dianalisis apakah ada titik di bagian kedua yang jarak komponen x,y, dan z kedua titik tersebut lebih kecil dari `d`. Jika iya, maka akan dihitung jarak *euclidean* nya, simpan sebagai `tempd`.
9. Apabila `tempd` lebih kecil dari `d`, maka nilai `d` akan digantikan dengan `tempd`, dan kedua pasang titik tersebut akan ditandai sebagai pasangan titik dengan jarak terkecil. Apabila tidak, maka jarak *euclidean* dan pasangan titik akan ditandai sama dengan yang diperoleh dari tahap sebelumnya.
10. Saat semua titik sudah diproses, maka `d` akan berisi jarak terkecil dari semua pasang titik, dan `pair` akan berisi pasangan titik yang memiliki jarak tersebut.

BAB 3
IMPLEMENTASI

1. Fungsi dan Prosedur yang Digunakan

Tabel 3.1.1 Tabel Fungsi dan Prosedur

Fungsi/Prosedur	Keterangan
Procedure quickSort(m,i,j)	Mengurutkan elemen-elemen dari m secara menaik berdasarkan nilai absisnya
Function Partition(m,i,j)	Mengembalikan nilai partisi untuk mekanisme quickSort
Function FindClosestPair(arrayOfPoint, pointCount,dimension,euc_count)	Mencari jarak terkecil dari pasangan titik pada himpunan titik arrayOfPoint dengan algoritma divide and conquer, dengan parameter banyak titik (pointCount) dan dimensi (dimension), sekaligus mengembalikan banyaknya pemanggilan fungsi euclidean distance.
Function euclidean_distance (point1, point2, dimension)	Menghitung nilai jarak *euclidean* antara point1 dan point2.
Function splitList(array,b)	Memecah array menjadi 2 list titik dengan jumlah titik yang sama,atau hanya berbeda 1.
Procedure printPoint(point)	Menampilkan titik.
Function bruteforce (arrayOfPoint, pointCount,dimension)	Mencari pasangan titik terdekat pada arrayOfPoint dan jaraknya dengan algoritma *brute force*.
Procedure show (PointList,pair)	Memvisualisasi semua titik pada PointList,dan menampilkan titik pada pair dengan warna yang berbeda.

2. Source Code

sorting.py

```
#Mengurutkan List berdasarkan nilai x nya
def Partition(m,i,j):
    pivot = m[j]
    p = i-1
    for q in range (i,j):
        if (m[q] <= pivot):
            p += 1
            temp = m[q]
            m[q] = m[p]
            m[p] = temp
    temp = m[p+1]
    m[p+1] = m[j]
```

```
    m[j] = temp
    return p+1

def quickSort(m,i,j):
    if(i<j):
        k = Partition(m,i,j)
        quickSort(m,i,k-1)
        quickSort(m,k,j)
    return
```

findClosestPair.py

```
from sorting import *

def FindClosestPair (arrayOfPoint, pointCount,dimension,euc_count):
    pair = []
    if (pointCount == 2):
        d = euclidean_distance(arrayOfPoint[0], arrayOfPoint[1],dimension)
        euc_count +=1
        pair.append(arrayOfPoint[0])
        pair.append(arrayOfPoint[1])

    elif (pointCount == 3): #Penanganan untuk banyak titik tidak 2^k
        d1 = euclidean_distance (arrayOfPoint[0], arrayOfPoint[1],dimension)
        d2 = euclidean_distance (arrayOfPoint[0], arrayOfPoint[2],dimension)
        d3 = euclidean_distance (arrayOfPoint[1], arrayOfPoint[2],dimension)
        euc_count += 3
        if (d1<=d2 and d1<=d3):
            d = d1
            pair.append(arrayOfPoint[0])
            pair.append(arrayOfPoint[1])
        elif (d2<=d1 and d2<=d3):
            d = d2
            pair.append(arrayOfPoint[0])
            pair.append(arrayOfPoint[2])
        else:
            d = d3
            pair.append(arrayOfPoint[1])
            pair.append(arrayOfPoint[2])

    else:
        k = pointCount//2
        arrayOfPoint1, arrayOfPoint2 = splitList(arrayOfPoint,pointCount)
        d1, pair1, euc_count = FindClosestPair (arrayOfPoint1,k,dimension,
        euc_count)
        d2, pair2, euc_count = FindClosestPair(arrayOfPoint2,pointCount-
        k,dimension,euc_count)
        if(d1<d2):
            d = d1
            pair = pair1
        else:
            d=d2
            pair = pair2

       #Mencari titik yang berada di dekat batas S1 dan S2
        evalmore1 = []
        evalmore2 = []
        xedge = (arrayOfPoint1[k-1][0]+arrayOfPoint2[0][0])/2

        for i in range(k):
```

```
            if (arrayOfPoint1[i][0]>=xedge - d):
                evalmore1.append(arrayOfPoint1[i])
        for i in range(len(arrayOfPoint)-k):
            if (arrayOfPoint2[i][0] <= xedge + d):
                evalmore2.append(arrayOfPoint2[i])

        for point1 in evalmore1:
            for point2 in evalmore2:
                if(check(point1,point2,dimension,d)):
                    pair3 = []
                    d3 = euclidean_distance(point1,point2,dimension)
                    euc_count += 1
                    if(d3<d):
                        d = d3
                        pair3.append(point1)
                        pair3.append(point2)
                        pair = pair3
    return d,pair,euc_count

def euclidean_distance(point1,point2,dimension):
    sum = 0
    for i in range (dimension):
        sum += (point1[i]-point2[i])**2
    return sum**(0.5)

def check(point1,point2,dimension,d):
    count = 0
    for i in range (dimension):
        if (abs(point1[i]-point2[i]) <d):
            count += 1
    return (count==dimension)

def splitList(array,b):
    k = b//2
    array1 = []
    array2 = []
    for i in range(k):
        array1.append(array[i])
    for i in range(k,b):
        array2.append(array[i])
    return array1, array2

def printPoint(point):
    print ("(“, end="")
    for i in range(len(point)-1):
        print(point[i], end="")
        print (",”, end="")
    print(point[len(point)-1], end="")
    print (")")
    return
```

bruteforce.py

```
from findClosestPair import euclidean_distance

def bruteforce(arrayOfPoint, pointCount,dimension):
    count =0
    pair = []
    d = 99999999999999999999
    for i in range(pointCount):
        for j in range(i+1, pointCount):
```

```
            pairnow= []
            dnow = euclidean_distance(arrayOfPoint[i], arrayOfPoint[j],
            dimension)
            count+=1
            #print(dnow)
            pairnow.append(arrayOfPoint[i])
            pairnow.append(arrayOfPoint[j])
            if(dnow<d):
                d = dnow
                pair = pairnow
    print ("Terdapat",count,"kali perhitungan euclidean distance")
    return d,pair
```

visualization.py

```
import matplotlib.pyplot as plt
def show (PointList,pair):
    x =[]
    y =[]
    z =[]
    xa = []
    ya = []
    za = []
    for point in PointList:
        if (point == pair[0] or point == pair[1]):
            xa.append(point[0])
            ya.append(point[1])
            za.append(point[2])
        else:
            x.append(point[0])
            y.append(point[1])
            z.append(point[2])
    fig = plt.figure(figsize = (50,50))
    ax = plt.axes(projection = "3d")
    ax.scatter3D(x,y,z,color="black")
#Menampilkan titik lain dengan warna hitam
    ax.scatter3D(xa,ya,za,color = "red")
#Menampilkan pasangan titik terdekat dengan warna merah
    plt.show()
```

main.py

```
import random
import time
from sorting import *
from findClosestPair import *
from bruteforce import *
from visualization import *

#Program untuk Mencari 2 Titik dengan Jarak Terpendek dari Daftar Titik yang
Dibangkitkan secara Acak

#Input dimensi dan generate titik random
R = input("Masukkan Dimensi (>=1): ")
R = float(R)
while (R//1 != R or R<1):
    print("Masukan tidak valid, silakan coba lagi!")
    R = input("Masukkan Dimensi (>=1): ")
    R = float(R)
R = int(R)
```

```
N = input("Masukkan Banyak Titik (>=2): ")
N = float(N)
while(N//1 != N or N<2):
    print("Masukan tidak valid, silahkan coba lagi")
    N = input("Masukkan Banyak Titik (>=2): ")
    N = float(N)
N = int(N)
PointList = [[(random.uniform(-1000000,1000000)) for j in range(R)] for i in
range (N)]

#Sorting elemen titik berdasarkan absis
quickSort(PointList,0,N-1)

#Pencarian dengan Divide and Conquer
euc_count=0                                                    #counter
start = time.time()                                            #inisialisasi waktu
d, pair,euc_count= FindClosestPair(PointList,N,R,euc_count) #mencari pasangan
titik terdekat
stop = time.time()                                             #waktu berhenti

print()
print("================================================================================
==============")
print("Pencarian dengan Divide and Conquer:")
print()
print("Pasangan titik terdekat: ")
printPoint(pair[0])                          #Menampilkan titik pertama
printPoint(pair[1])                          #Menampilkan titik kedua
print("Dengan jarak",d)                      #Menampilkan jarak kedua titik
print()
#Menampilkan berapa kali pemanggilan euclidean distance
print("Terdapat",euc_count, "kali perhitungan euclidean distance")
print("Waktu eksekusi: ", (stop-start)*1000, "ms")     #Menampilkan waktu
print("Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3
GenuineIntel")
print("================================================================================
==============")

#Pencarian dengan Brute Force
print("Tampilkan Solusi dengan Brute Force? (y/n)")
n=input()
if (n=='y'):
     print("================================================================================
     ====================")
    start = time.time()                          #inisialisasi waktu
    print("Pencarian dengan Brute Force:")
    d2,pair2 = bruteforce(PointList,N,R)         #Perhitungan dengan brute force
    stop = time.time()                           #waktu berhenti
    print("Pasangan titik: ")
    printPoint(pair2[0])                         #Menampilkan titik pertama
    printPoint(pair2[1])                         #Menampilkan titik kedua
    print("Dengan jarak",d2)                     #Menampilkan jarak
    print()
    print("Waktu eksekusi: ", (stop-start)*1000, "ms")     #Menampilkan waktu
    print("Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3
    GenuineIntel")

print("================================================================================
==============")
if (R == 3):
    print("Tampilkan visualisasi data? (y/n)")
```

```
    n=input()
    if (n=='y'):
        show(PointList,pair)
print("Terima kasih sudah menggunakan program ini!")
```

3. Analisis Kompleksitas Algoritma

Pada algoritma *divide and conquer* yang diterapkan pada fungsi findClosestPair, terdapat 2 kali pemanggilan ulang fungsi dengan parameter setengah dari list awal. Pada fungsi terdapat dua kali iterasi untuk setiap elemen list titik (n^2) saat membandingkan jarak titik pada list 1 dan list 2. Walaupun iterasi dilakukan dua kali, namun jumlah titik yang diperiksa pada setiap kasus tergolong konstan, yaitu maksimal hanya 6 titik, sehingga proses tersebut bisa digolongkan sebagai kasus O(n). Kompleksitas algoritma findClosestPair dapat dituliskan sebagai berikut:

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + cn, n > 3 \\ a, \quad n = 2, n = 3 \end{cases}$$

Dengan Teorema Master, dan dengan a=2, b=2, d=1, dan dengan mengetahui bahwa jika $a = b^d$, diperoleh kompleksitas algoritma tersebut adalah $T(n) = O(n^d \log n) = O(n \log n)$. Algoritma ini lebih mangkus daripada *brute force* yang memiliki kompleksitas $O(n^2)$.

BAB 4
PENGUJIAN

1. Tampilan Awal Program

```
PS C:\Users\User\Documents\GitHub\Tucil2_13521042> python -u "c:\Users\User\Documents\GitHub\Tucil2_13521042\src\main.py"
Masukkan Dimensi:
```

Gambar 4.1.1 Tampilan Awal Program

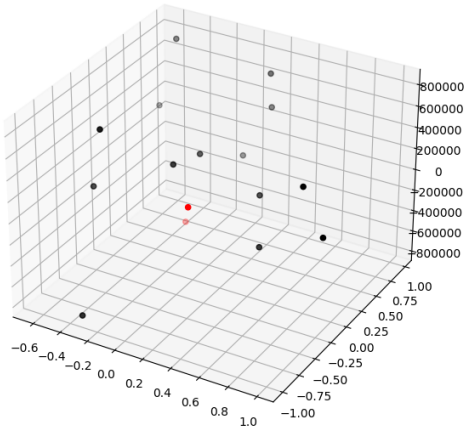
2. Pengujian Program dengan Uji Coba 16 Titik

```
Masukkan Dimensi: 3
Masukkan Banyak Titik: 16

================================================================================
Pencarian dengan Divide and Conquer:

Pasangan titik terdekat:
(13080.011865928653,-126763.57520086307,-246379.12953780615)
(95443.6630054568,-257183.5619662326,-3924.8334422250045)
Dengan jarak 287362.540481735

Terdapat 12 kali perhitungan euclidean distance
Waktu eksekusi: 0.9999275207519531 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
```

Gambar 4.2.1 Pengujian Program dengan 16 Titik

```
================================================================================
Tampilkan Solusi dengan Brute Force? (y/n)
y
================================================================================
Pencarian dengan Brute Force:
Terdapat 120 kali perhitungan euclidean distance
Pasangan titik:
(13080.011865928653,-126763.57520086307,-246379.12953780615)
(95443.6630054568,-257183.5619662326,-3924.8334422250045)
Dengan jarak 287362.540481735

Waktu eksekusi:  66.00809097290039 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
```

Gambar 4.2.2 Perbandingan dengan Brute Force (16 Titik)



Gambar 4.2.3 Visualisasi 16 Titik dan Pasangan Titik Terdekat

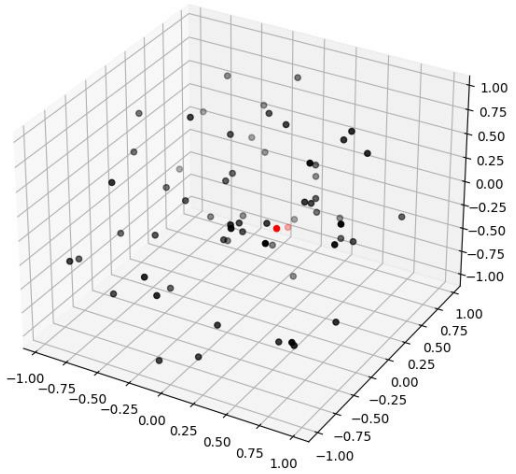
3. Pengujian Program dengan Uji Coba 64 Titik

```
Masukkan Dimensi (>=1): 3
Masukkan Banyak Titik (>=2): 64

================================================================================
Pencarian dengan Divide and Conquer:

Pasangan titik terdekat:
(-236229.1831166012,864134.68629588,-870424.0581454124)
(-173447.89884694875,918151.644582842,-869311.141149235)
Dengan jarak 82828.4976403815

Terdapat 64 kali perhitungan euclidean distance
Waktu eksekusi: 2.0008087158203125 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
```

Gambar 4.3.1 Pengujian Program dengan 64 Titik

```
================================================================================
Tampilkan Solusi dengan Brute Force? (y/n)
y
================================================================================
Pencarian dengan Brute Force:
Terdapat 2016 kali perhitungan euclidean distance
Pasangan titik:
(-236229.1831166012,864134.68629588,-870424.0581454124)
(-173447.89884694875,918151.644582842,-869311.141149235)
Dengan jarak 82828.4976403815

Waktu eksekusi:  17.99631118774414 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
```

Gambar 4.3.2 Perbandingan dengan Algoritma Brute Force (64 Titik)



Gambar 4.3.3 Visualisasi 64 Titik dan Pasangan Titik Terdekat

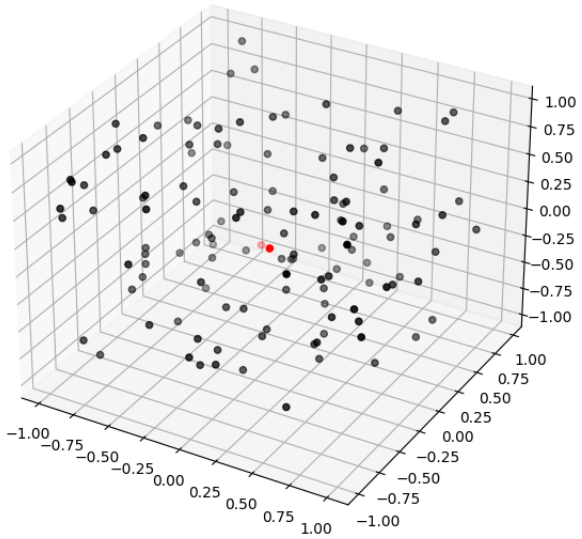
4. Pengujian Program dengan Uji Coba 128 Titik

```
Masukkan Dimensi (>=1): 3
Masukkan Banyak Titik (>=2): 128

================================================================================
Pencarian dengan Divide and Conquer:

Pasangan titik terdekat:
(256085.27553170826,-599083.8344332952,465535.3212960183)
(310185.635453627,-596753.2512045705,454640.7865814797)
Dengan jarak 55235.599467313186

Terdapat 118 kali perhitungan euclidean distance
Waktu eksekusi: 6.00123405456543 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
```

Gambar 4.4.1 Pengujian Program dengan Uji Coba 128 Titik

```
================================================================================
Tampilkan Solusi dengan Brute Force? (y/n)
y
================================================================================
Pencarian dengan Brute Force:
Terdapat 8128 kali perhitungan euclidean distance
Pasangan titik:
(256085.27553170826,-599083.8344332952,465535.3212960183)
(310185.635453627,-596753.2512045705,454640.7865814797)
Dengan jarak 55235.599467313186

Waktu eksekusi:  68.53485107421875 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
```

Gambar 4.4.2 Perbandingan dengan Algoritma Brute Force (128 Titik)



Gambar 4.4.3 Visualisasi 128 Titik dan Pasangan Titik Terdekat

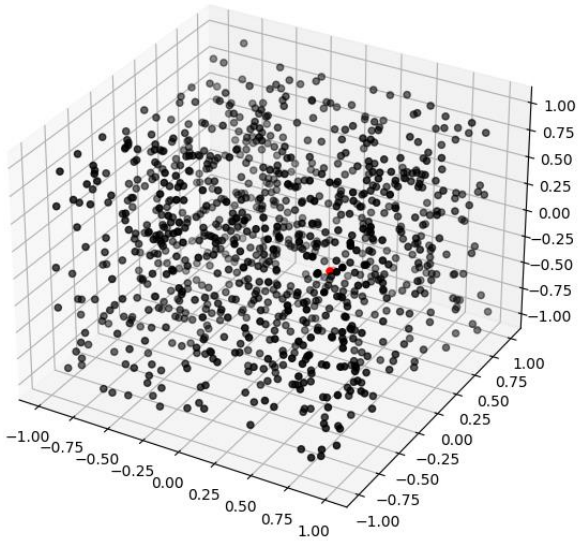
5. Pengujian Program dengan Uji Coba 1000 Titik

```
Masukkan Dimensi (>=1): 3
Masukkan Banyak Titik (>=2): 1000

================================================================================
Pencarian dengan Divide and Conquer:

Pasangan titik terdekat:
(551070.9325291109,-297012.31939742784,161476.32139710896)
(567933.6823145994,-282587.4266132014,165238.51398731233)
Dengan jarak 22507.42000430527

Terdapat 983 kali perhitungan euclidean distance
Waktu eksekusi: 42.9682731628418 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
```

Gambar 4.5.1 Pengujian Program dengan Uji Coba 1000 Titik

```
================================================================================
Tampilkan Solusi dengan Brute Force? (y/n)
y
================================================================================
Pencarian dengan Brute Force:
Terdapat 499500 kali perhitungan euclidean distance
Pasangan titik:
(551070.9325291109,-297012.31939742784,161476.32139710896)
(567933.6823145994,-282587.4266132014,165238.51398731233)
Dengan jarak 22507.42000430527

Waktu eksekusi:  1360.4145050048828 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
```

Gambar 4.5.2 Perbandingan dengan Algoritma Brute Force



Gambar 4.5.3 Visualisasi 1000 Titik dan Pasangan Titik Terdekat

6. Pengujian untuk dimensi selain 3 (tiga)

a. Dimensi dua

```
PS C:\Users\User\Documents\GitHub\Tucil2_13521042> python -u "src\main.py"
Masukkan Dimensi (>=1): 2
Masukkan Banyak Titik (>=2): 128

================================================================================
Pencarian dengan Divide and Conquer:

Pasangan titik terdekat:
(-356020.9922172965,531086.9370453076)
(-348447.3251321709,532221.0845092225)
Dengan jarak 7658.114884631807

Terdapat 115 kali perhitungan euclidean distance
Waktu eksekusi: 1.9981861114501953 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
Tampilkan Solusi dengan Brute Force? (y/n)
y
================================================================================
Pencarian dengan Brute Force:
Terdapat 8128 kali perhitungan euclidean distance
Pasangan titik:
(-356020.9922172965,531086.9370453076)
(-348447.3251321709,532221.0845092225)
Dengan jarak 7658.114884631807

Waktu eksekusi:  28.998851776123047 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
Terima kasih sudah menggunakan program ini!
```

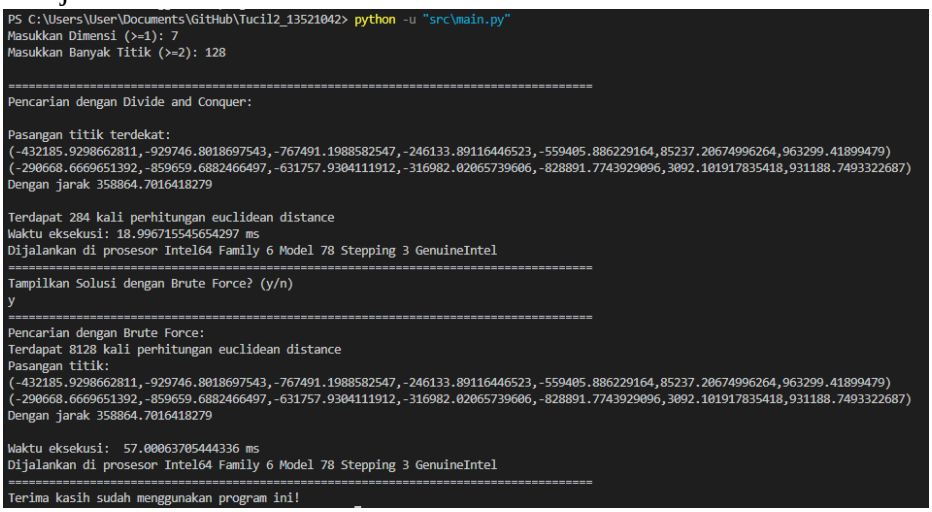
Gambar 4.6.1 Dua Dimensi

b. Dimensi empat

```
PS C:\Users\User\Documents\GitHub\Tucil2_13521042> python -u "src\main.py"
Masukkan Dimensi (>=1): 4
Masukkan Banyak Titik (>=2): 128

================================================================================
Pencarian dengan Divide and Conquer:

Pasangan titik terdekat:
(-36884.978611709084,-844091.3139561797,-527352.1594488465,769432.2828134354)
(-19140.045842346502,-778312.6824615072,-548997.3559280657,855497.4749977167)
Dengan jarak 111881.37841769023

Terdapat 122 kali perhitungan euclidean distance
Waktu eksekusi: 11.043548583984375 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
Tampilkan Solusi dengan Brute Force? (y/n)
y
================================================================================
Pencarian dengan Brute Force:
Terdapat 8128 kali perhitungan euclidean distance
Pasangan titik:
(-36884.978611709084,-844091.3139561797,-527352.1594488465,769432.2828134354)
(-19140.045842346502,-778312.6824615072,-548997.3559280657,855497.4749977167)
Dengan jarak 111881.37841769023

Waktu eksekusi:  52.00004577636719 ms
Dijalankan di prosesor Intel64 Family 6 Model 78 Stepping 3 GenuineIntel
================================================================================
Terima kasih sudah menggunakan program ini!
```

Gambar 4.6.2 Vektor pada R^4

c. Dimensi tujuh



Gambar 4.6.3 Vektor dalam R^7

BAB 5
KESIMPULAN

Dari pengujian yang sudah dilakukan, dapat dilihat bahwa solusi yang ditampilkan oleh algoritma *divide and conquer* sama seperti solusi yang diberikan oleh algoritma *brute force*. Solusi yang diberikan oleh *brute force* sudah tentu benar karena memeriksa semua kemungkinan. Oleh karena itu, bisa disimpulkan bahwa algoritma *divide and conquer* dapat digunakan untuk mencari pasangan titik dengan jarak terdekat. Bisa dilihat juga bahwa waktu eksekusi algoritma *divide and conquer* lebih cepat daripada algoritma *brute force*. Artinya, algoritma *divide and conquer* lebih mangkus daripada algoritma *brute force* dalam menangani persoalan ini.

DAFTAR PUSTAKA

Munir, Rinaldi. 2021. “Algoritma Divide and Conquer (2021) Bagian 1”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf, diakses pada 25 Februari 2023, pukul 22.24 WIB.

Munir, Rinaldi. 2021. “Algoritma Divide and Conquer (2021) Bagian 2”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf, diakses pada 25 Februari 2023, pukul 22.24 WIB.

LAMPIRAN

Link Repository GitHub:

https://github.com/kevinjohn01/Tucil2_13521042

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	✓	