

Tugas Kecil 2 IF2211 Strategi Algoritma
Algoritma *Divide and Conquer* untuk Mencari Pasangan
Titik Terdekat pada Ruang *Euclidian*

Disusun oleh:

Juan Christopher Santoso	13521116
Brigita Tri Carolina	13521156



Institut Teknologi Bandung
Teknik Informatika
2023

Daftar Isi

1	Deskripsi Masalah dan Algoritma	2
1.1	Algoritma <i>Divide and Conquer</i>	2
1.2	Mencari Pasangan Titik Terdekat pada Bidang 3 Dimensi (<i>Closest Pair</i>)	2
1.3	Algoritma Penyelesaian Mencari Pasangan Titik Terdekat pada N-Dimensi dengan <i>Divide and Conquer</i>	3
2	Implementasi Algoritma Dalam Bahasa Python	5
2.1	main.py	5
2.2	calculate.py	5
2.3	dnc.py	5
2.4	bf.py	6
2.5	in_out.py	6
2.6	visualize.py	7
2.7	globalCount.py	7
3	Source Code Program	8
3.1	main.py	8
3.2	calculate.py	9
3.3	bf.py	11
3.4	dnc.py	12
3.5	globalCounts.py	14
3.6	in_out.py	15
3.7	visualize.py	17
4	Masukan dan Luaran Program	19
5	Lampiran	28
5.1	Repository Program	28
5.2	<i>Progress Table</i>	28
6	Daftar Pustaka	29

1 Deskripsi Masalah dan Algoritma

1.1 Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* merupakan suatu algoritma yang secara rekursif membagi suatu permasalahan ke dalam dua bagian sub-permasalahan dengan jenis yang sama atau mirip satu sama lain, sampai pada akhirnya permasalahan menjadi cukup sederhana untuk diselesaikan secara langsung. Solusi untuk sub-masalah kemudian digabungkan (*combine*) untuk memberikan solusi untuk permasalahan aslinya.

Singkatnya algoritma ini terbagi menjadi tiga tahap:

1. *Divide*: membagi persoalan menjadi beberapa sub-persoalan yang memiliki kemiripan namun **berukuran lebih kecil** (idealnya hampir sama).
2. *Conquer*: menyelesaikan masing-masing sub-persoalan secara langsung jika sudah cukup sederhana atau secara rekursif jika masih berukuran besar.
3. *Combine*: menggabungkan solusi dari masing-masing sub-persoalan sehingga membentuk solusi untuk permasalahan semula.

1.2 Mencari Pasangan Titik Terdekat pada Bidang 3 Dimensi (*Closest Pair*)

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 $P1 = (x1, y1, z1)$ dan $P2 = (x2, y2, z2)$ dihitung dengan rumus Euclidean berikut:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad (1)$$

Penyelesaian permasalahan mencari pasangan titik terdekat pada program ini dibuat dalam Bahasa Python dengan menerapkan Algoritma *Divide and Conquer* untuk penyelesaiannya dan Algoritma Brute Force sebagai pembandingnya.

Masukan program adalah sebagai berikut.

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

- Dimensi: dimensi yang diinginkan pada *Euclidian Space*
- *Boundary*: batasan daerah ketika mem-visualize titik-titik pada grafik 3D

Luaran program adalah sebagai berikut.

- Sepasang titik yang jaraknya terdekat dan nilai jaraknya
- Banyaknya operasi perhitungan rumus Euclidian
- Waktu riil dalam detik
- Penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya.

1.3 Algoritma Penyelesaian Mencari Pasangan Titik Terdekat pada N-Dimensi dengan *Divide and Conquer*

Pada penyelesaian mencari titik terdekat pada N-Dimensi digunakan algoritma dengan pendekatan *Divide and Conquer*. Langkah-langkah penyelesaiannya dengan menggunakan metode *Divide and Conquer* adalah sebagai berikut:

1. Program menerima input banyaknya titik yang ingin di-generate, dimensi yang diinginkan, dan *boundary* ketika memvisualisasi data.
2. Ketiga data disimpan pada array data untuk digunakan pada fungsi dnc.
3. Langkah penyelesaian dengan *Divide and Conquer*:
 - (a) Pertama-tama titik-titik yang telah ter-generate diurutkan berdasarkan nilai absisnya (x).
 - (b) Kemudian titik-titik dibagi ke dalam dua himpunan sehingga kedua himpunan memiliki banyak titik yang idealnya hampir sama.
 - (c) Setelah itu karena fungsi dnc adalah sebuah fungsi rekursif dan fungsi pada program mengembalikan jarak titik terdekat maka dibentuk dua basis. Basis pertama adalah ketika jumlah titik dua maka fungsi akan langsung mengembalikan jarak kedua titik tersebut. Jarak antara kedua titik dihitung menggunakan fungsi *getDistance* yang dapat menghitung jarak kedua titik pada N-Dimensi. Basis kedua adalah ketika jumlah titik adalah tiga (dalam kasus banyaknya titik ganjil diperlukan basis ini) maka fungsi akan membandingkan ketiga titik secara langsung dan langsung meng-*assign* jarak dua titik terdekat.

- (d) Maka setelah basis terbentuk, langkah selanjutnya adalah kasus ketika banyak titik lebih dari dua dan tiga. Di sini diterapkan algoritma *Divide and Conquer* yang membagi persoalan menjadi dua secara rekursif terus-menerus hingga persoalan mencapai basis dan dapat diselesaikan secara langsung.
- (e) Setelah jarak terdekat di antara dua wilayah persoalan tadi sudah ditemukan, diperlukan fungsi akhir untuk menentukan jarak terdekat paling *final*. Pada fungsi ini himpunan titik-titik diseleksi dengan titik yang berada pada *range* $X[\text{amount}/2] \pm$ jarak terdekat yang telah ditemukan tadi dimasukkan pada *array of points* baru.
- (f) Kemudian *array of points* tersebut diurutkan berdasarkan nilai ordinat (y) yang menaik.
- (g) Langkah akhir adalah menghitung jarak setiap pasang titik pada *array of points* tadi dengan menggunakan *loop* sebanyak $N-1$ kali dan bandingkan apakah jaraknya lebih kecil dari jarak yang sudah ditemukan dengan fungsi rekursif tadi.

2 Implementasi Algoritma Dalam Bahasa Python

Program ini dibuat dalam bahasa Pemrograman Python. Adapun struktur dari program pada folder *src* yang terdiri dari *calculate.py*, *dnc.py*, *in_out.py*, *visualize.py*, dan *main.py* sebagai driver utama program.

2.1 *main.py*

File merupakan driver utama program berisi alur utama program dan pemanggilan fungsi-fungsi untuk menyelesaikan permasalahan utama.

2.2 *calculate.py*

File berisi hitungan matematika yang digunakan dalam program.

Fungsi	Deskripsi
<code>getDistance(arr1, arr2)</code>	Fungsi menghasilkan jarak antara dua titik pada N-Dimensi
<code>mergerSort(arr, n)</code>	Fungsi mengembalikan array yang telah di-sorting menggunakan algoritma <i>mergesort</i>
<code>mergeArr(arr1, arr2, elmt)</code>	Fungsi digunakan pada <i>merging</i> hasil dari <code>mergerSort</code>
<code>splitArrBot(arr, n)</code>	Fungsi mengembalikan array berisi setengah elemen arr bagian bawah
<code>splitArrTop(arr, n)</code>	Fungsi mengembalikan array berisi setengah elemen arr bagian atas
<code>generateNumber(length, boundaries)</code>	Fungsi men-generate random number yang akan digunakan sebagai titik-titik pada bidang N-Dimensi
<code>startTime()</code>	Fungsi mengembalikan waktu dan digunakan pada saat memulai algoritma perhitungan
<code>stopTime()</code>	Fungsi mengembalikan waktu dan digunakan pada akhir algoritma perhitungan
<code>converToSeconds(micros)</code>	Fungsi mengembalikan <i>micros</i> (ms) menjadi <i>second</i>

2.3 *dnc.py*

File berisi fungsi-fungsi yang menyusun algoritma *Divide and Conquer* dalam penyelesaian untuk mencari pasangan titik terdekat pada bidang N-Dimensi.

Fungsi	Deskripsi
notQualified(point1, point2, d)	Fungsi mengembalikan <i>true</i> apabila point tidak berada dalam <i>range</i> d
getClosestPairOf3(p1, p2, p3)	Fungsi mengembalikan jarak terdekat antara ketiga titik p1, p2, p3
getSolutionDnC(arr, amount)	Fungsi mengembalikan tuple dua titik terdekat pada bidang N-Dimensi
thirdCase(arr, tupleP, amount)	Fungsi memeriksa kembali titik-titik di sekitar tengah-tengah X untuk mengembalikan <i>tuple</i> titik terdekat paling <i>final</i> .

2.4 bf.py

File ini berisi implementasi penyelesaian permasalahan menggunakan algoritma *Brute Force*.

Fungsi	Deskripsi
getSolutionBF(arr)	Fungsi untuk menyelesaikan permasalahan <i>closest pair</i> menggunakan algoritma <i>brute force</i> . Keluaran dari fungsi ini adalah list yang menampung index dari pasangan titik <i>closest pair</i> .

2.5 in_out.py

File berisi fungsi-fungsi mengatur masukan dan luaran program.

Fungsi	Deskripsi
displayTitle(title)	Fungsi menampilkan <i>title</i> pada terminal
askUserInput()	Fungsi menerima <i>input</i> dari <i>user</i> berupa banyak titik, dimensi, dan boundary
validateUserInput()	Fungsi mengembalikan nilai <i>true</i> ketika <i>input user</i> sesuai
displayArr()	Fungsi menampilkan array solusi pada terminal

2.6 visualize.py

File berisi fungsi yang digunakan untuk memvisualisasi titik pada bidang 3D.

Fungsi	Deskripsi
visualize(arr, solutions, amount, dimension, boundary)	Fungsi berisi algoritma untuk memvisualisasi titik pada grafik 3D dengan boundary sebagai batasan grafik

2.7 globalCount.py

File berisi *attribute count* untuk banyaknya operasi getDistanceEuclidian yang dilakukan. File menyimpan *variable* berupa *EuclidianCountDnC* dan *EuclidianCountBF*.

Fungsi	Deskripsi
addCountsDnC(n)	Fungsi melakukan <i>increment</i> pada <i>EuclidianCountDnC</i>
addCountsBF(n)	Fungsi melakukan <i>increment</i> pada <i>EuclidianCountBF</i>
getCountsDnC(n)	Fungsi mengembalikan <i>EuclidianCountDnC</i>
getCountsBF(n)	Fungsi mengembalikan <i>EuclidianCountBF</i>

3 Source Code Program

3.1 main.py

```
from in_out import *
from calculate import *
from bf import *
from dnc import *
from visualize import *
from globalCounts import *

# Contains main algorithm
# Ask User Input
displayTitle("User Input")
data = validateUserInput()
print()

pointsArray = []
displayTitle("Generating Number")
for i in range(data[0]):
    pointsArray.append(generateNumber(data[1], data[2]))
print("Random points are successfully generated.")
print()

#Calculating (Divide and Conquer Algorithm)
displayTitle("Divide and Conquer Algorithm")
dncStart = startTime()
dncSolution = getSolutionDnC(pointsArray, data[0])
dncStop = stopTime()

displaySubTitle("Euclidean Distance Counts")
print(getCountsDnC())
displaySubTitle("Execution Time")
print(dncStop - dncStart)
displaySubTitle("Amount of Solution")
print(len(dncSolution))
displaySubTitle("Nearest Distance")
print(getDistance(dncSolution[0][0], dncSolution[0][1]))
displaySubTitle("Points")
print()
displayPoints(dncSolution)
print()
#Calculating (Brute Force Algorithm)
```

```

displayTitle("Brute Force Algorithm")
bfStart = startTime()
bfSolution = getSolutionBF(pointsArray)
bfStop = stopTime()

displaySubTitle("Euclidean Distance Counts")
print(getCountsBF())
displaySubTitle("Execution Time")
print(bfStop - bfStart)
displaySubTitle("Amount of Solution")
print(len(bfSolution))
displaySubTitle("Nearest Distance")
print(getDistance(pointsArray[bfSolution[0][0]], pointsArray[
    bfSolution[0][1]]))
displaySubTitle("Points")
print()
displayPointByIndex(pointsArray, bfSolution)
print()

#visualizing if the dots are in 3D
displayTitle("Visualizing")
if (data[1] == 3 or data[1] == 2):
    print("Do you want to visualize the dots? (y/n)")
    answer = input(">> ")
    if (answer == "y"):
        visualize(pointsArray, dncSolution, data[0], data[1], data
[2])
    else :
        print("The program has been stopped.")
else :
    print("It cannot be visualized.")

```

3.2 calculate.py

```

import random
import time
import numpy as np

# Random Number
def generateNumber(length, boundaries):

```

```

    # Generate random number (float) and inserting it into a tuple
    # Generated random number will be in range of (-boundaries < x
    < boundaries)
    container = []
    for i in range (length):
        container.append(random.uniform(-boundaries, boundaries))
    return tuple(container)

# Time
def startTime():
    return time.time()

def stopTime():
    return time.time()

def convertToSeconds(micros):
    return micros* (10 ** 6)

# Distance calculation
def getDistance(arr1, arr2):
    # Determine the distance of two points
    res = 0
    for i in range(len(arr1)):
        res += (arr2[i] - arr1[i])**2
    return res**(0.5)

# =====
# Contains calculation of Divide and Conquer Algorithm

# Data Sorting
def mergeArr(arr1, arr2, elmt):
    # Merging two sorted array
    new = [0 for i in range(len(arr1) + len(arr2))]
    idx1 = 0
    idx2 = 0
    idx = 0
    while (idx1 < len(arr1) or idx2 < len(arr2)):
        if (idx1 == len(arr1)):
            new[idx] = arr2[idx2]
            idx2 += 1
            idx += 1
        elif (idx2 == len(arr2)):
            new[idx] = arr1[idx1]
            idx1 += 1

```

```

        idx += 1
    else:
        if (arr1[idx1][elmt] < arr2[idx2][elmt]):
            new[idx] = arr1[idx1]
            idx1 += 1
            idx += 1
        else :
            new[idx] = arr2[idx2]
            idx2 += 1
            idx += 1
    return new

def splitArrBot(arr, n):
    new = []
    for i in range(n):
        new.append(arr[i])
    return new

def splitArrTop(arr, n):
    new = []
    for i in range (n, len(arr)):
        new.append(arr[i])
    return new

def mergeSort(arr, elmt):
    # Will be sorted by the first index of each tuple
    if (len(arr) > 1):
        half = int(len(arr)/2)
        arr1 = mergeSort(splitArrBot(arr, half), elmt)
        arr2 = mergeSort(splitArrTop(arr, half), elmt)
        return mergeArr(arr1, arr2, elmt)
    else :
        return arr

```

3.3 bf.py

```

from calculate import *
from globalCounts import *
# =====
# Calculation for Brute Force Algorithm

```

```

def getSolutionBF(arr):
    arrSolution = []
    for i in range(len(arr)):
        for j in range(i+1, len(arr)):
            # Define the first pair of elements as the min tuple
            if (i == 0 and j == 1):
                # Min tuple will store the index of the shortest
pair
                minTuple = (i, j)
                arrSolution.append(minTuple)
            else :
                d1 = getDistance(arr[i], arr[j])
                d2 = getDistance(arr[minTuple[0]], arr[minTuple
[1]])

                addCountsBF(2)
                if ( d1 < d2):
                    arrSolution.clear()
                    minTuple = (i, j)
                    arrSolution.append(minTuple)
                elif (d1 == d2):

                    minTuple = (i,j)
                    arrSolution.append(minTuple)
    return arrSolution

```

3.4 dnc.py

```

from calculate import *
from globalCounts import *

# =====
# Calculation for Divide and Conquer Algorithm
def notQualified(point1, point2, d):
    for i in range (len(point1)):
        if (abs(point1[i] - point2[i]) > d):
            return True
    return False

def getClosestPairof3(p1, p2, p3):

```

```

d1 = getDistance(p1, p2)
d2 = getDistance(p2, p3)
d3 = getDistance(p3, p1)

if (min(d1,d2,d3) == d1):
    return [(p1,p2)]
elif (min(d1,d2,d3) == d2):
    return [(p2,p3)]
else :
    return [(p1,p3)]

def getSolutionDnC(arr, amount):
    arr = mergeSort(arr,0)
    if (amount == 2):
        return [(arr[0], arr[1])]
    elif (amount == 3):
        addCountsDnC(3)
        return getClosestPairof3(arr[0], arr[1], arr[2])
    else :
        # split array into 2
        newAmount = amount // 2
        arr1 = splitArrBot(arr, newAmount)
        arr2 = splitArrTop(arr, newAmount)

        # get closest points of left side and right side
        ptuple1 = getSolutionDnC(arr1, newAmount)
        ptuple2 = getSolutionDnC(arr2, newAmount)

        d1 = getDistance(ptuple1[0][0], ptuple1[0][1])
        d2 = getDistance(ptuple2[0][0], ptuple2[0][1])
        addCountsDnC(2)
        if ( d1 < d2 ):
            ptuple2.clear()
        elif (d1 == d2):
            ptuple1.append(ptuple2[0])
            ptuple2.clear()
        else :
            ptuple1.clear()
            for i in range(len(ptuple2)):
                ptuple1.append(ptuple2[i])
            ptuple2.clear()

        #shortest points will be stored in ptuple1

```

```

        return thirdCase(arr, ptuple1, amount)
def thirdCase(arr, tupleP, amount):
    # calculate minimal distance
    d = getDistance(tupleP[0][0], tupleP[0][1])
    addCountsDnC(1)
    # default answer will be the points from tupleP
    # will be changed if there is any other shorter pair
    result = []
    if (amount == 2):
        return tupleP
    else :
        points = []
        newAmount = amount // 2

        # Insert points within d range
        for i in range(amount):
            if (arr[i][0] >= arr[newAmount][0] - int(d) + 1
                or arr[newAmount][0] + int(d) + 1):
                points.append(arr[i])

        # Sort based on the second element (y-axis)
        points = mergeSort(points, 1)
        for i in range(len(points)):
            for j in range(i+1, len(points)):
                # case the point is outside the d range

                if (notQualified(points[i], points[j], d)):
                    continue
                else :
                    d1 = getDistance(points[i], points[j])
                    addCountsDnC(1)
                    if (d1 < d):
                        result.clear()
                        result.append((points[i], points[j]))
                        d = getDistance(points[i], points[j])
                        addCountsDnC(1)
                    elif (d1 == d):
                        result.append((points[i], points[j]))

    return result

```

3.5 globalCounts.py

```

EuclideanCountDnC = 0
EuclideanCountBF = 0

def addCountsDnC(n):
    global EuclideanCountDnC
    EuclideanCountDnC += n

def addCountsBF(n):
    global EuclideanCountBF
    EuclideanCountBF += n

def getCountsDnC():
    global EuclideanCountDnC
    return EuclideanCountDnC

def getCountsBF():
    global EuclideanCountBF
    return EuclideanCountBF

```

3.6 in_out.py

```

# Contain input and output algorithm

# Title Printing
def displayTitle(title):
    # Displaying title
    length = len(title)
    if ((56 - length) % 2 == 0):
        fspace = int((56 - length)/2)
        bspace = fspace
    else:
        fspace = int((56 - length + 1)/2)
        bspace = fspace - 1
    print("
=====
")
    print("====", end="") ; print(" "*fspace, end="")
    print(title, end="")
    print(" "*bspace, end=""); print("====")

```



```

    print("
=====
")
def displaySubTitle(subtitle):
    length = len(subtitle)
    space = 25 - length
    print(subtitle, space*" ", end=": ")

# User Input
def askUserInput():
    # Return tuple of three value, amount of points, dimension, and
    boundaries
    try:
        amount = input("Amount of points you want to insert: ")
        dimension = input("Dimension of Euclidean Space you want to
use: ")
        boundaries = input("Boundaries of Euclidean Space you want
to use: ")
        amount = int(amount)
        dimension = int(dimension)
        boundaries = int(boundaries)
        return (amount, dimension, boundaries)
    except:
        return (None, None, None)

def validateUserInput():
    data = askUserInput()
    valid = False
    while (not valid):
        valid = True
        if (data[0] == None):
            print(" => (Some of) the answers you just inserted
cannot be converted into integer or float.")
            valid = False
        elif (data[0] < 2 or data[1] < 1):
            print(" => The minimum amount of points are 2 and the
minimum dimensions of Euclidean Plane are 2")
            valid = False

        if (not valid):
            print(" => Please insert again.")
            data = askUserInput()

```

```

        return data
# Displaying Arr
def displayArr(arr):
    for i in range(len(arr)):
        print(arr[i])

def displayPointByIndex(arrPoint, arrIndex):
    for i in range(len(arrIndex)):
        print(i+1, end=". ")
        print(arrPoint[arrIndex[i][0]])
        print("    ", end="")
        print(arrPoint[arrIndex[i][1]])

def displayPoints(arrPoints):
    for i in range(len(arrPoints)):
        print(i+1, end=". ")
        print(arrPoints[i][0])
        print("    ", end="")
        print(arrPoints[i][1])

```

3.7 visualize.py

```

import matplotlib.pyplot as plt

def visualize(arr, solutions, amount, dimension, boundary):
    if (dimension == 3):
        fig = plt.figure()
        ax = plt.axes(projection='3d')
        ax.set_title('Closest Pair of Dots in 3D')
        dots = []
        for i in range (boundary + 1):
            dots.append(i)
        print(dots)
        ax.scatter(dots, dots, dots, c='white', alpha=0)
        for i in range (amount):
            x = arr[i][0]
            y = arr[i][1]
            z = arr[i][2]
            ax.scatter(x, y, z, c='black')
        for i in range(len(solutions)):

```

```

        for j in range(2):
            x = solutions[i][j][0]
            y = solutions[i][j][1]
            z = solutions[i][j][2]
            ax.scatter(x,y,z, c="red")
plt.show()
else :
    fig = plt.figure()
    ax = plt.axes()
    ax.set_title('Closest Pair of Dots in 2D')
    dots = []
    for i in range (boundary + 1):
        dots.append(i)
    print(dots)
    ax.scatter(dots, dots, c='white', alpha=0)
    for i in range (amount):
        x = arr[i][0]
        y = arr[i][1]
        ax.scatter(x, y, c='black')
    for i in range(len(solutions)):
        for j in range(2):
            x = solutions[i][j][0]
            y = solutions[i][j][1]
            ax.scatter(x,y, c="red")
plt.show()

```

4 Masukan dan Luaran Program

Input	<div data-bbox="603 398 1398 683"><pre>===== ==== User Input ==== ===== Amount of points you want to insert: 16 Dimension of Euclidean Space you want to use: 3 Boundaries of Euclidean Space you want to use: 100000 ===== ==== Generating Number ==== ===== Random points are successfully generated.</pre></div> <p>Gambar 1. Pada bidang 3 dimensi dengan banyak <i>point</i> yang di-generate adalah 16</p>
Output	<div data-bbox="612 884 1385 1404"><pre>===== ==== Divide and Conquer Algorithm ==== ===== Euclidean Distance Counts : 42 Execution Time : 0.0015435218811035156 Amount of Solution : 1 Nearest Distance : 32472.884611929665 Points : 1. (-60193.50582755527, 66987.97923704228, 34589.32633871012) (-28697.563753905924, 70013.4285747864, 41892.786081720056) ===== ==== Brute Force Algorithm ==== ===== Euclidean Distance Counts : 238 Execution Time : 0.0010204315185546875 Amount of Solution : 1 Nearest Distance : 32472.884611929665 Points : 1. (-60193.50582755527, 66987.97923704228, 34589.32633871012) (-28697.563753905924, 70013.4285747864, 41892.786081720056)</pre></div>

Input

```
=====
====                        User Input                        ====
=====
Amount of points you want to insert: 64
Dimension of Euclidean Space you want to use: 3
Boundaries of Euclidean Space you want to use: 10000

=====
====                        Generating Number                  ====
=====
Random points are successfully generated.
```

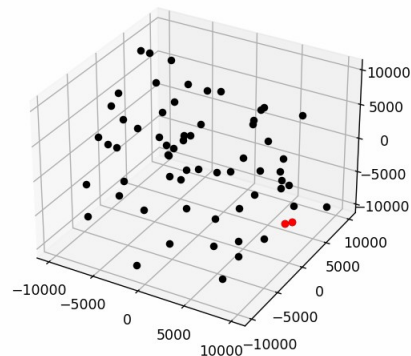
Gambar 2. Pada bidang 3 dimensi dengan banyak *point* yang di-generate adalah 64

Output

```
=====
====                        Divide and Conquer Algorithm      ====
=====
Euclidean Distance Counts : 184
Execution Time             : 0.004998207092285156
Amount of Solution         : 1
Nearest Distance           : 746.839761530193
Points                     :
1. (7199.223361009255, 4413.5291754978825, -9714.27427340589)
   (7800.9805986455285, 4797.377288857913, -9494.459619151943)

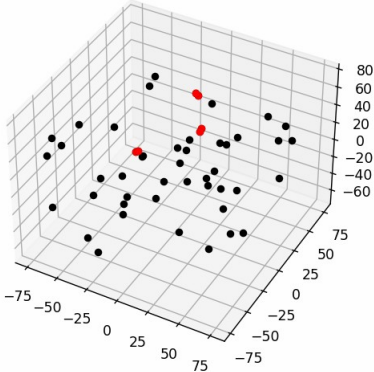
=====
====                        Brute Force Algorithm             ====
=====
Euclidean Distance Counts : 4030
Execution Time             : 0.005032062530517578
Amount of Solution         : 1
Nearest Distance           : 746.839761530193
Points                     :
1. (7199.223361009255, 4413.5291754978825, -9714.27427340589)
   (7800.9805986455285, 4797.377288857913, -9494.459619151943)
```

Closest Pair of Dots in 3D



<p>Input</p>	<div data-bbox="604 383 1396 663" data-label="Code-Block"> <pre> ===== User Input ===== Amount of points you want to insert: 128 Dimension of Euclidean Space you want to use: 3 Boundaries of Euclidean Space you want to use: 10000 ===== Generating Number ===== Random points are successfully generated. </pre> </div> <p>Gambar 3. Pada bidang 3 dimensi dengan banyak <i>point</i> yang di-generate adalah 128</p>
<p>Output</p>	<div data-bbox="612 864 1385 1384" data-label="Code-Block"> <pre> ===== Divide and Conquer Algorithm ===== Euclidean Distance Counts : 360 Execution Time : 0.008508682250976562 Amount of Solution : 1 Nearest Distance : 366.13531716507606 Points : 1. (4818.379097107449, -7312.833129260712, -568.4026009249737) (4899.61722289697, -7182.817139451845, -235.91007501228887) ===== Brute Force Algorithm ===== Euclidean Distance Counts : 16254 Execution Time : 0.01557612419128418 Amount of Solution : 1 Nearest Distance : 366.13531716507606 Points : 1. (4899.61722289697, -7182.817139451845, -235.91007501228887) (4818.379097107449, -7312.833129260712, -568.4026009249737) </pre> </div> <div data-bbox="882 1516 1145 1796" data-label="Figure"> <p>Closest Pair of Dots in 3D</p> </div>

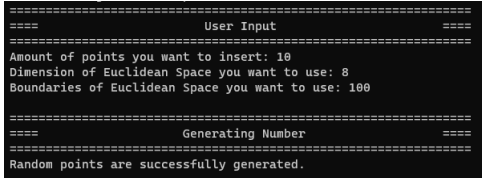
<p>Input</p>	<div data-bbox="608 327 1394 616" data-label="Code-Block"> <pre> ===== ==== User Input ==== ===== Amount of points you want to insert: 1000 Dimension of Euclidean Space you want to use: 3 Boundaries of Euclidean Space you want to use: 100000 ===== ==== Generating Number ==== ===== Random points are successfully generated. </pre> </div> <p>Gambar 4. Pada bidang 3 dimensi dengan banyak <i>point</i> yang di-generate adalah 1000</p>
<p>Output</p>	<div data-bbox="616 819 1386 1337" data-label="Code-Block"> <pre> ===== ==== Divide and Conquer Algorithm ==== ===== Euclidean Distance Counts : 2289 Execution Time : 0.40615296363830566 Amount of Solution : 1 Nearest Distance : 787.5503841821504 Points : 1. (9952.34329284662, 80366.718563414, -14115.252010983022) (10063.247413576668, 80573.69996662394, -14866.97974251605) ===== ==== Brute Force Algorithm ==== ===== Euclidean Distance Counts : 998998 Execution Time : 0.992283821105957 Amount of Solution : 1 Nearest Distance : 787.5503841821504 Points : 1. (9952.34329284662, 80366.718563414, -14115.252010983022) (10063.247413576668, 80573.69996662394, -14866.97974251605) </pre> </div> <div data-bbox="810 1476 1185 1861" data-label="Figure"> <p>Closest Pair of Dots in 3D</p> </div>

Input	<pre> ===== ==== User Input ==== ===== Amount of points you want to insert: 50 Dimension of Euclidean Space you want to use: 3 Boundaries of Euclidean Space you want to use: 70 ===== ==== Generating Number ==== ===== Random points are successfully generated. </pre> <p>Gambar 5. Pada bidang 3 dimensi dengan banyak <i>point</i> yang di-generate adalah 50. Namun solusi yang didapatkan lebih dari satu pasang titik</p>
Output	<pre> ===== ==== Divide and Conquer Algorithm ==== ===== Euclidean Distance Counts : 124 Execution Time : 0.003454923629760742 Amount of Solution : 3 Nearest Distance : 2.8284271247461903 Points : 1. (-20, -17, 26) (-22, -17, 24) 2. (10, 20, 30) (10, 22, 32) 3. (-8, 50, 38) (-10, 50, 40) ===== ==== Brute Force Algorithm ==== ===== Euclidean Distance Counts : 3078 Execution Time : 0.004000186920166016 Amount of Solution : 3 Nearest Distance : 2.8284271247461903 Points : 1. (10, 20, 30) (10, 22, 32) 2. (-10, 50, 40) (-8, 50, 38) 3. (-20, -17, 26) (-22, -17, 24) </pre> <p>Closest Pair of Dots in 3D</p> 

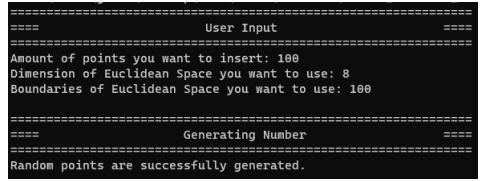
<p>Input</p>	<div data-bbox="683 672 1319 896"> <pre> ===== ==== User Input ==== ===== Amount of points you want to insert: 10 Dimension of Euclidean Space you want to use: 4 Boundaries of Euclidean Space you want to use: 100 ===== ==== Generating Number ==== ===== Random points are successfully generated. </pre> </div> <p>Gambar 6. Pada bidang 4 dimensi dengan banyak <i>point</i> yang di-generate adalah 10</p>
<p>Output</p>	<div data-bbox="616 1099 1386 1581"> <pre> ===== ==== Divide and Conquer Algorithm ==== ===== Euclidean Distance Counts : 18 Execution Time : 0.0 Amount of Solution : 1 Nearest Distance : 55.56891825899091 Points : 1. (-9.067800502725973, -1.3172145746583368, 56.4047652907648, 30.93637841086013) (-15.786677984151083, 18.40822046593007, 7.027167879323912, 45.617027177083145) ===== ==== Brute Force Algorithm ==== ===== Euclidean Distance Counts : 88 Execution Time : 0.0014793872833251953 Amount of Solution : 1 Nearest Distance : 55.56891825899091 Points : 1. (-9.067800502725973, -1.3172145746583368, 56.4047652907648, 30.93637841086013) (-15.786677984151083, 18.40822046593007, 7.027167879323912, 45.617027177083145) ===== ==== Visualizing ==== ===== It cannot be visualized. </pre> </div>

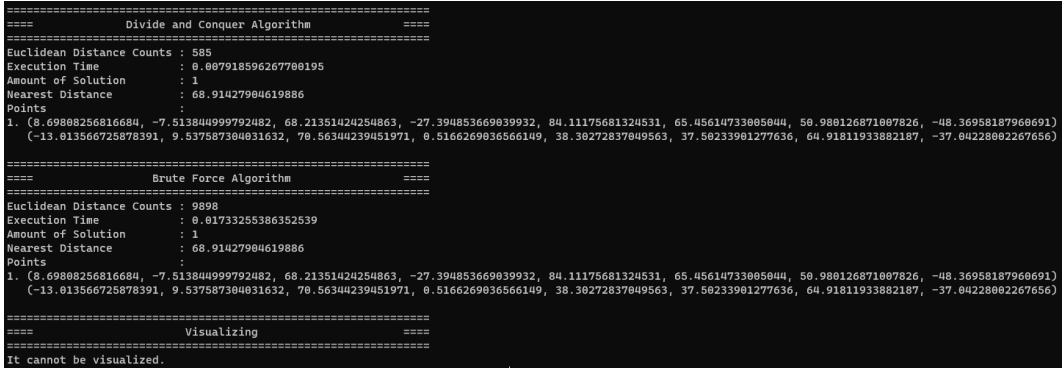
Input	<pre> ===== ==== User Input ===== ===== Amount of points you want to insert: 100 Dimension of Euclidean Space you want to use: 4 Boundaries of Euclidean Space you want to use: 100 ===== ==== Generating Number ===== ===== Random points are successfully generated. </pre> <p>Gambar 7. Pada bidang 4 dimensi dengan banyak <i>point</i> yang di-generate adalah 100</p>
Output	<pre> ===== ==== Divide and Conquer Algorithm ===== ===== Euclidean Distance Counts : 279 Execution Time : 0.005961179733276367 Amount of Solution : 1 Nearest Distance : 15.554295742017842 Points : 1. (65.90424765948308, -47.510594441532916, -53.52707975259135, -11.982373272522096) (67.28744216904605, -45.915603163624354, -55.49288773000529, 3.3020793116629363) ===== ==== Brute Force Algorithm ===== ===== Euclidean Distance Counts : 9898 Execution Time : 0.010562658309936523 Amount of Solution : 1 Nearest Distance : 15.554295742017842 Points : 1. (65.90424765948308, -47.510594441532916, -53.52707975259135, -11.982373272522096) (67.28744216904605, -45.915603163624354, -55.49288773000529, 3.3020793116629363) ===== ==== Visualizing ===== ===== It cannot be visualized. </pre>

Input	<pre> ===== ==== User Input ==== ===== Amount of points you want to insert: 1000 Dimension of Euclidean Space you want to use: 4 Boundaries of Euclidean Space you want to use: 100 ===== ==== Generating Number ==== ===== Random points are successfully generated. </pre> <p>Gambar 8. Pada bidang 4 dimensi dengan banyak <i>point</i> yang di-generate adalah 1000</p>
Output	<pre> ===== ==== Divide and Conquer Algorithm ==== ===== Euclidean Distance Counts : 2422 Execution Time : 0.300396203994751 Amount of Solution : 1 Nearest Distance : 2.6407872247564512 Points : 1. (-14.274482035772863, -84.89134701044651, 80.36203390069164, -31.74954251666928) (-15.428633501732648, -83.723535767969, 79.66850944498503, -29.800970657245117) ===== ==== Brute Force Algorithm ==== ===== Euclidean Distance Counts : 998998 Execution Time : 0.9264826774597168 Amount of Solution : 1 Nearest Distance : 2.6407872247564512 Points : 1. (-15.428633501732648, -83.723535767969, 79.66850944498503, -29.800970657245117) (-14.274482035772863, -84.89134701044651, 80.36203390069164, -31.74954251666928) ===== ==== Visualizing ==== ===== It cannot be visualized. </pre>

Input	
Output	<p>Gambar 9. Pada bidang 8 dimensi dengan banyak <i>point</i> yang di-generate adalah 10</p>

Output	
--------	---

Input	
Output	<p>Gambar 10. Pada bidang 8 dimensi dengan banyak <i>point</i> yang di-generate adalah 100</p>

Output	
--------	--

5 Lampiran

5.1 Repository Program

Repository program dapat diakses melalui:

- Github: https://github.com/Gulilil/Tucil2_13521116_13521156

5.2 *Progress Table*

Poin	Ya	Tidak
Program berhasil dikompilasi tanpa kesalahan	✓	
Program berhasil <i>running</i>	✓	
Program dapat menerima masukan dan menuliskan luaran	✓	
Luaran program sudah benar (solusi <i>close pair</i> benar	✓	
Bonus 1 dikerjakan	✓	
Bonus 2 dikerjakan	✓	

6 Daftar Pustaka

- Munir, Rinaldi. *Algoritma Divide and Conquer (Bagian 1)*. 2023. URL: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf) (visited on 02/27/2023).
- *Algoritma Divide and Conquer (Bagian 2)*. 2023. URL: [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) (visited on 02/27/2023).
- *Algoritma Divide and Conquer (Bagian 3)*. 2023. URL: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian3.pdf) (visited on 02/27/2023).