

IF2211 Strategi Algoritma  
**IMPLEMENTASI ALGORITMA DIVIDE AND CONQUER  
UNTUK PENCARIAN TITIK TERDEKAT DALAM DIMENSI KE-N**

**Laporan Tugas Kecil II**

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma  
pada Semester II (dua) Tahun Akademik 2022/2023



**Oleh**

**Kenneth Ezekiel Suprantonni      13521089**

**Alisha Listya Wardhani      13521171**

**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
BANDUNG  
2023**

## DAFTAR ISI

<b>BAB I DESKRIPSI MASALAH.....</b>	<b>3</b>
<b>BAB II TEORI SINGKAT.....</b>	<b>3</b>
<b>BAB III RANCANGAN DAN IMPLEMENTASI PROGRAM.....</b>	<b>6</b>
3.1. Rancangan Algoritma .....	6
3.2. Generalisasi Algoritma dalam Vektor $Rn$ .....	7
3.3. Implementasi Program dalam Bahasa Python .....	8
3.4. Source Code Program .....	10
<b>BAB IV EKSPERIMEN DAN ANALISIS .....</b>	<b>15</b>
4.1. Eksperimen Program.....	15
4.2. Perbandingan dengan Brute Force .....	16
4.3. Masukkan dan Keluaran Program.....	16
<b>BAB V KESIMPULAN DAN SARAN.....</b>	<b>22</b>
5.1. Kesimpulan .....	22
5.2. Saran .....	22
<b>LAMPIRAN.....</b>	<b>23</b>
Pranala Repositori Github.....	24
Checklist Fitur.....	24

## BAB I

### DESKRIPSI MASALAH

Pada tugas ini, permasalahan yang akan dibahas merupakan masalah dengan kedekatan titik-titik  $N$  dalam ruang tiga dimensi. Tujuan tugas ini adalah untuk menentukan sepasang titik di dalam himpunan titik-titik tersebut yang jarak *euclidean*-nya terdekat satu sama lain dengan algoritma *Divide and Conquer*. Setiap titik  $P$  di dalam ruang dinyatakan dengan koordinat  $P = (x, y, z)$ . 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.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad (1.1.)$$

Adapun dalam tugas ini juga akan dilakukan generalisasi algoritma tersebut sehingga dapat menyelesaikan permasalahan kedekatan titik-titik  $N$  dalam ruang- $k$  Euclidean. Pada tugas kecil ini, digunakan algoritma *brute force* sebagai tolak ukur yang akan dibandingkan secara kompleksitas dan validitas penyelesaian.

## BAB II

### TEORI SINGKAT

#### 2.1. Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* merupakan algoritma dengan pendekatan rekursif yang memanfaatkan pembagian persoalan menjadi upapersoalan yang lebih kecil, menyelesaikannya secara rekursif dan menggabungkan solusinya untuk mendapatkan solusi pada masalah utama.

Berikut merupakan skema umum dalam algoritma *Divide and Conquer*.

**Procedure** DIVIDEandCONQUER(input P: problem, n: integer)

{ Menyelesaikan persoalan P dengan algoritma divide and conquer

Masukan: masukan persoalan P berukuran n

Luaran: solusi dari persoalan semula }

**Deklarasi**

r : integer

**Algoritma**

if  $n \leq n_0$  then {ukuran persoalan P sudah cukup kecil }

SOLVE persoalan P yang berukuran n

else

DIVIDE menjadi r upa-persoalan,  $P_1, P_2, \dots, P_r$ , yang masing-masing berukuran  $n_1, n_2, \dots, n_r$

for masing-masing  $P_1, P_2, \dots, P_r$ , do

DIVIDEandCONQUER( $P_i, n_i$ )

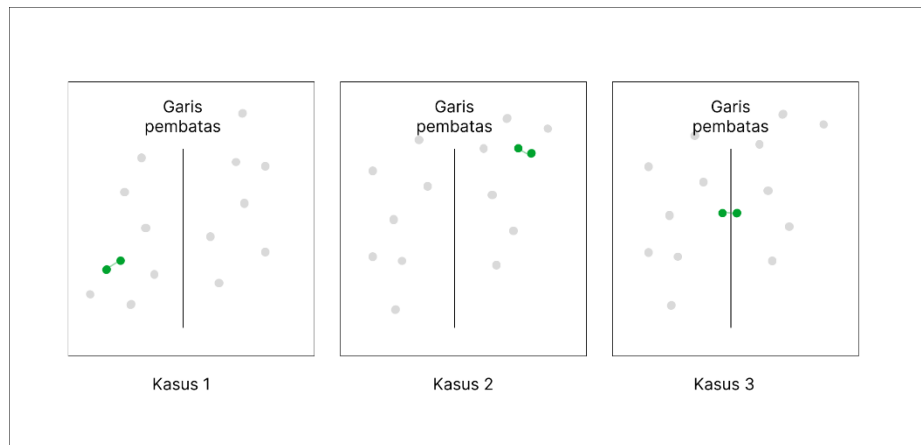
endfor

COMBINE solusi dari  $P_1, P_2, \dots, P_r$  menjadi persoalan semula

endif

## 2.2. Pencarian Titik Terdekat

Permasalahan pencarian pasangan titik terdekat umumnya diselesaikan di dalam 2 Dimensi, dimana untuk mendapatkan pasangan titik terdekat secara *brute force*, hanya diperlukan untuk menghitung jarak dari suatu titik ke titik-titik lainnya. Pendekatan *divide and conquer* didalam permasalahan ini adalah untuk membagi himpunan titik-titik menjadi 2 bagian, lalu menghitung titik terdekat untuk setiap bagian secara rekursif.



Terdapat 3 kasus utama dalam pengaplikasian algoritma *divide and conquer* di permasalahan ini, yaitu:

1. Pasangan titik terdekat berada di himpunan bagian kiri
2. Pasangan titik terdekat berada di himpunan bagian kanan
3. Pasangan titik terdekat terpisahkan di himpunan kiri dan kanan

Untuk kasus pertama dan kedua, dapat ditangani dengan mudah dengan kasus-kasus basis untuk perhitungan tiap sisinya, dan untuk penggabungannya hanya perlu untuk membandingkan jarak terkecil saja. Tetapi untuk kasus ketiga, perlu pendekatan yang lebih heuristik, dimana diambil titik-titik yang dekat dengan garis pembagi, definisi dekat disini adalah jarak terkecil yang didapatkan dari penggabungan hasil perhitungan kasus 1 dan 2, dan membandingkan jarak-jarak antar titik-titik yang diambil tersebut, sehingga, akan didapatkan jarak terkecil untuk kasus ke-3.

Terdapat beberapa optimasi untuk perbandingan jarak dari titik-titik yang sudah diambil tersebut, dimana perbandingan titik hanya perlu dilakukan jika titik tersebut terdapat pada himpunan (pada kasus 1 dan 2) yang berbeda, dan ditambah juga hanya perlu membandingkan titik yang jarak di sumbu y dan z nya (atau sumbu k untuk k-dimensi) lebih kecil dari jarak yang didapatkan dari perhitungan sebelumnya (kasus 1 dan 2). Optimasi ini dijelaskan secara lebih detail pada teori *sparsity condition*.

Berikut adalah skema umum dari algoritma pemecahan masalah pencarian pasangan terdekat:

**Procedure** GetClosestPair(input A: Larik point, n: integer)

{ Menyelesaikan persoalan pencarian pasangan titik terdekat dengan algoritma divide and conquer

Masukan: masukan larik point A berukuran n

Luaran: solusi dari persoalan pencarian titik terdekat }

**Deklarasi**

r : integer

### Algoritma

```
if  $n = 1$  then {Basis 1}
    return NULL
else if  $n = 2$  then {Basis 2}
    return Jarak( $A_1, A_2$ )
else if  $n = 3$  then {Basis 3}
    return Min(Jarak( $A_1, A_2$ ), Jarak( $A_1, A_3$ ), Jarak( $A_2, A_3$ ))
else
    DIVIDE menjadi 2 upa-larik  $P_1, P_2$  yang masing-masing berukuran  $n_1 = FLOOR\left(\frac{n}{2}\right), n_2 = n - n_1$ 
    for masing-masing  $P_1, P_2$  do
        GetClosestPair( $P_i, n_i$ )
    endfor
    COMBINE solusi dari  $P_1, P_2$  menjadi jarak terdekat ( $\delta$ )
    ambil titik-titik yang berada sejauh  $\delta$  di kiri dan kanan garis tengah pembagi sebagai  $P_3, P_4$ 
    bandingkan jarak titik-titik dari  $P_3$  dan  $P_4$  yang memiliki jarak y (dan z) lebih kecil dari  $\delta$ 
    return hasil jarak titik terkecil
endif
```

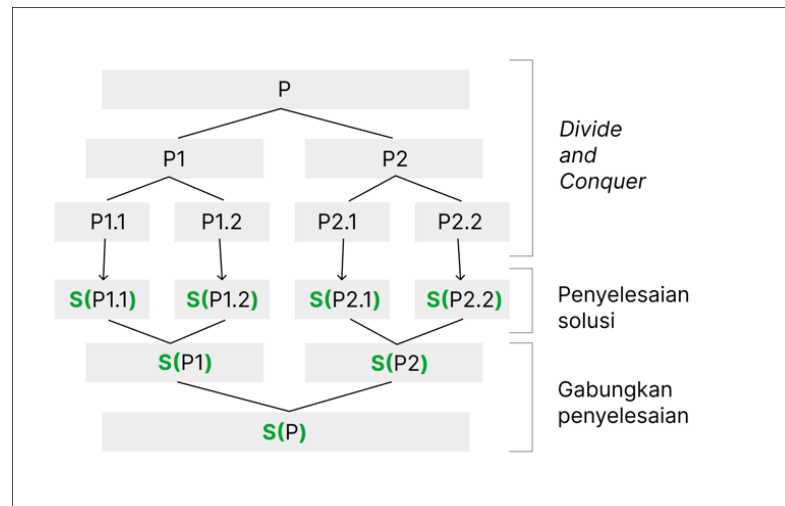
Sparsity Condition yang terdapat pada pencarian titik terdekat yaitu jika terdapat sebuah kubus dengan panjang sisi  $2\delta$  mengandung  $O(1)$  titik S. Perlu diperhatikan bahwa permasalahan utama belum tentu memiliki kondisi ini.

## BAB III

### RANCANGAN DAN IMPLEMENTASI PROGRAM

#### 3.1. Rancangan Algoritma

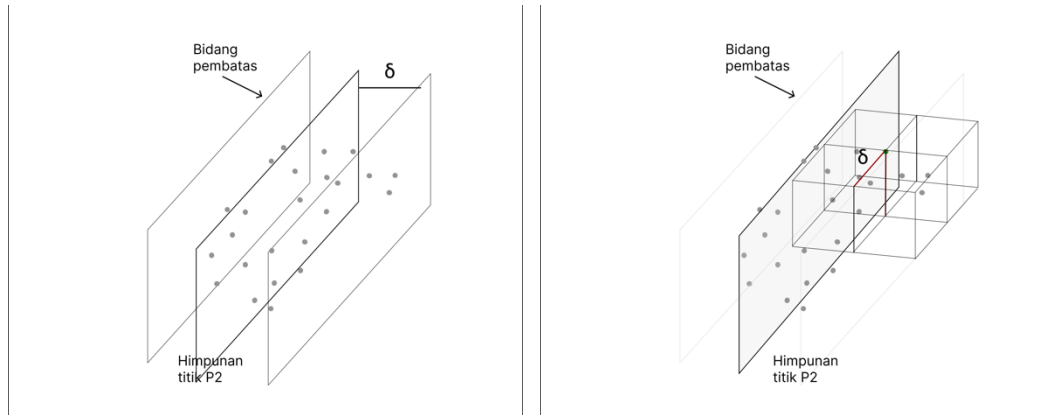
Algoritma *divide and conquer* untuk penyelesaian permasalahan pasangan titik terdekat dalam 3 dimensi sangat dekat dengan penyelesaiannya dalam 2 dimensi, sehingga kerangka rancangan awalnya sangat dekat pula. Perbedaannya hanya terletak pada rumus perhitungan jarak *Euclidean*-nya, dan kasus dimana pasangan titik terdekat dipisahkan oleh garis tengah pembagi himpunan. Rumus perhitungan jarak hanya perlu memperhitungkan dimensi tambahan, menjadi rumus perhitungan jarak *Euclidean* untuk 3 dimensi, sama halnya dengan penanganan kasus untuk 3 dimensi, perubahan yang dibutuhkan hanya terdapat pada *sparsity condition*, dimana titik-titik yang dibatasi bukan hanya dalam sumbu  $y$ , tetapi dalam sumbu  $z$  juga, sehingga membentuk sebuah kisi-kisi kubus.



Gambar 3.1.1. Langkah *divide and conquer*

Berikut adalah langkah-per-langkah dari algoritma *divide and conquer* yang diterapkan:

1. Kumpulkan semua titik menjadi suatu himpunan  $P_1$
2. Bagi himpunan  $P_1$  menjadi dua Himpunan  $P_2$  dan  $P_3$  dari sebuah titik tengah himpunan
3. Bagi Himpunan  $P_2$  dan  $P_3$  masing-masing menjadi dua himpunan yang lebih kecil
4. Saat Himpunan hasil pembagian sudah berisikan 1, 2, atau 3 titik, dapat dihitung jaraknya secara iteratif
5. Gabungkan hasil perhitungan dari himpunan-himpunan yang lebih kecil menjadi solusi dari himpunan yang lebih besar (mencari *local minimum* untuk *range* yang lebih besar)
6. Setelah semua himpunan digabungkan, menjadi  $P_2$  dan  $P_3$ , gabungkan hasil  $P_2$  dan  $P_3$  menjadi hasil dari  $P_1$  (*local minimum* menjadi *global minimum*)



Gambar 3.2.1. Segmentasi Dalam Titik-titik

Untuk perhitungan jarak dalam himpunan-himpunan kecil, dapat

1. Hitung jarak 3 dimensi untuk himpunan kiri dan kanan, hasil jarak terkecil adalah  $\delta$
2. Hitung jarak 3 dimensi untuk titik-titik antar himpunan kiri dan kanan yang berada sejauh  $\delta$  dari garis tengah pembagi, dan berjarak maksimal sejauh  $\delta$  di sumbu y dan z (karena tidak mungkin terdapat jarak 3 dimensi yang lebih kecil dari  $\delta$  yang muncul dari y dan z dengan jarak yang lebih besar dari  $\delta$ , karena jarak x sudah dibatasi maksimal sejauh  $\delta$ )
3. Gabungkan kedua perhitungan menjadi solusi yang diharapkan untuk himpunan tersebut

### 3.2. Generalisasi Algoritma dalam Vektor $R^n$

Permasalahan pencarian pasangan titik terdekat dapat diperluas juga menjadi n-dimensi, yang akan menunjukkan perubahan dari perluasan penyelesaian masalah dari 2 dimensi menjadi 3 dimensi. Hanya saja, sekarang, titik-titik perlu dipandang sebagai vektor di ruang  $R^n$ , dan tidak ada cara untuk memvisualisasikan hasilnya diluar dari 3 dimensi dan 2 dimensi. Terlebih dari itu, perubahan yang terjadi sama seperti perluasan 2 dimensi menjadi 3 dimensi, dimana perhitungan jarak sekarang perlu memperhitungkan jarak di dimensi perluasan, dan juga *sparsity condition* akan membatasi titik-titik pada dimensi perluasan tersebut juga.

### 3.3. Implementasi Program dalam Bahasa Python

Dalam Tugas Kecil II ini, terdapat folder yang berisi lima modul pembantu dan satu file utama. Berikut merupakan rincian serta deskripsi dari setiap file tersebut.

#### 3.3.1. Bruteforce.py

Pada file bruteforce, diimplementasikan sebuah fungsi sebagai validasi kebenaran algoritma *Divide and Conquer* dan menjadi tolak ukur perbandingan algoritma.

Nama Fungsi / Prosedur	Deskripsi
bruteforce	Fungsi yang menerima input vector pada n dimensi dan mengembalikan pasangan titik terdekat serta nilai jarak terdekat tersebut. Fungsi ini menggunakan algoritma bruteforce yang mengecek satu persatu titik

#### 6.3.2. Sorting.py

Pada file ini merupakan implementasi fungsi pengurutan karena pengurutan tidak memperbolehkan memakai *library*. Algoritma yang digunakan merupakan algoritma QuickSort (*Divide and Conquer*).

Nama Fungsi / Prosedur	Deskripsi
partitioning	Fungsi untuk membagi larik vector menjadi larik-larik yang lebih kecil. Fungsi ini menerima input vector, jumlah dimensi, batas bawah, dan batas atas dan mengeluarkan pointer untuk membagi larik menjadi dua larik yang lebih kecil.
quickSort	Prosedur ini merupakan prosedur pengurutan yang menggunakan prinsip divide and conquer menggunakan algoritma rekursif.

#### 3.3.3. Visualizer.py

Pada file ini, dilakukan visualisasi khusus untuk vector-vektor pada bidang yang dapat divisualisasi (1D, 2D, ataupun 3D). Fungsi pada file ini menggunakan library matplotlib.

Nama Fungsi / Prosedur	Deskripsi
Visualize1DResult	Prosedur yang menerima vector satu dimensi dan pasangan titik terdekat, kemudian memvisualisasikan titik-titik tersebut serta menghubungkan titik-titik dengan jarak terdekat.
Visualize2DResult	Prosedur yang menerima vector dua dimensi dan pasangan titik terdekat, kemudian memvisualisasikan titik-titik tersebut serta menghubungkan titik-titik dengan jarak terdekat.
Visualize3DResult	Prosedur yang menerima vector tiga dimensi dan pasangan titik terdekat, kemudian memvisualisasikan titik-titik tersebut serta menghubungkan titik-titik dengan jarak terdekat.



#### 3.3.4. DivideAndConquer.py

File ini berisi algoritma utama yang akan dites dalam tugas kecil. Algoritma yang digunakan menggunakan konsep *divide and conquer*.

Nama Fungsi / Prosedur	Deskripsi
<code>getDistanceBetweenTwoPoints</code>	Fungsi yang menerima input 2 vektor di n dimensi dan mengembalikan jarak antara kedua vektor tersebut menggunakan rumus jarak <i>euclidean</i>
<code>getClosestPair</code>	Fungsi yang menerima input larik vektor di n dimensi dan jumlah vektornya dan mengembalikan jarak terdekat dan indeks dari vektor yang berkoresponden dengan jarak tersebut dalam larik vektor utama berdasarkan algoritma <i>divide and conquer</i>

#### 3.3.5. InputHandler.py (Class)

Kelas ini berisi objek vector yang diolah pada program.

Atribut	Deskripsi
<code>num</code>	Jumlah titik yang terdapat pada objek
<code>dimension</code>	Nomor dimensi objek tersebut berada
<code>size</code>	Besar threshold pada objek
<code>vecArr</code>	Larik vector berisi titik-titik yang dimiliki objek
Nama Fungsi / Prosedur	Deskripsi
<code>__init__</code>	Mengkonstruksi objek InputHandler dengan parameter jumlah vektor, jumlah dimensi, batas maksimal di sumbu koordinat, Boolean penggunaan decimal dengan nilai <i>default</i> True, Boolean random dengan nilai <i>default</i> True, dan larik vektor masukan dengan nilai <i>default</i> larik kosong
<code>randomizeVector</code>	Mengacak dan membuat vektor-vektor yang akan ditempatkan dalam atribut larik vektor objek, pengacakan dilakukan dengan pengambilan sampel acak dari distribusi normal
<code>printVectors</code>	Menunjukkan atribut larik vektor dari objek

#### 3.3.6. main.py

Nama Fungsi / Prosedur	Deskripsi
Program utama	Menetapkan alur program utama

### 3.4. Source Code Program

Berikut merupakan struktur dari program pada Tugas Kecil II Pencarian Titik Terdekat

```
README
├── doc
│   └── Tucil2_13521089_13521171
├── input
│   └── test
├── src
│   ├── Modules
│   │   ├── Bruteforce.py
│   │   ├── DivideandConquer.py
│   │   ├── InputHandler.py
│   │   ├── Sorting.py
│   │   └── Visualizer.py
│   └── main.py
└── test
    └── visualizer.png
```

```
Bruteforce.py
import math
import numpy

n = 0

def bruteforce(vectors: numpy.array):
    global n
    # print(vectors.vecArr.shape[0])
    closest = (numpy.sqrt(numpy.sum(numpy.square(vectors[1][:] - vectors[0][:]))))

    # VecBar = [x, y, z, ... ]
    vecBar = []
    idxpair = numpy.array([])

    for i in range(0, vectors.shape[0]):
        for j in range(1, vectors.shape[0]):
            # Iterate according to dimentions
            if (i != j):
                for k in range(0, vectors.shape[1]):
                    temp = 0
                    vecBar.append(vectors[j][k] - vectors[i][k])
                    for p in range(len(vecBar)):
                        temp += pow(vecBar[p],2)

                vecBar = []

                # Check if value is smaller than closest value
                val = math.sqrt(temp)

                n += 1
                if (val < closest):
                    closest = val
                    idxpair = numpy.array([i, j])
                elif (val == closest) and ([j, i] not in idxpair):
                    idxpair = numpy.append(idxpair, [i, j])

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

DivideAndConquer.py

```
import numpy

n = 0
# ===== #
# DIVIDE AND CONQUER
# CLOSEST DISTANCE ALGORITHM

def getDistanceBetweenTwoPoints(A : numpy.array, B : numpy.array):
    global n
    n += 1
    # return numpy.linalg.norm(A - B)
    # sum of squares
    squared_sum = numpy.sum(numpy.square(A - B))

    # Square rooting it gives the euclidean distance
    return numpy.sqrt(squared_sum)

def getClosestPair(vectors : numpy.array, n : int):
    closest : float
    idxpair = numpy.array([])
    if (n == 1):
        # print("No Closest Pair for one point!")
        pass
    elif (n == 2):
        # print("Pair")
        closest = getDistanceBetweenTwoPoints(vectors[0], vectors[1])
        idxpair = numpy.array([0, 1])
    elif (n == 3):
        # print("Triplet")
        closest = getDistanceBetweenTwoPoints(vectors[1], vectors[2])
        idxpair = numpy.array([1, 2])
        for i in range(0, 3):
            if (i != 0):
                temp = getDistanceBetweenTwoPoints(vectors[0], vectors[i])
                if temp < closest:
                    idxpair = numpy.array([0, i])
                    closest = temp
    else:
        # case 1 : smallest pair is in the same side
        # divide the array
        n_div = int(n / 2)
        left = vectors[0:n_div]
        right = vectors[n_div:n]

        # split into left and right
        leftidxpair, leftclosest = getClosestPair(left, n_div)
        rightidxpair, rightclosest = getClosestPair(right, n - n_div)

        # select the closest between the left and right subdivision
        if leftclosest < rightclosest:
            closest = leftclosest
            idxpair = leftidxpair
        elif rightclosest < leftclosest:
            closest = rightclosest
            idxpair = rightidxpair[0:rightidxpair.size] + n_div
        else:
            closest = leftclosest # or right, it's the same
            idxpair = numpy.append(leftidxpair, rightidxpair[0:rightidxpair.size] + n_div)

        # case 2 : smallest pair is on a separate subdivision
        # use closest as delta
        # x0 is the middle point of division
        x0 = ((vectors[n_div-1][0] + vectors[n_div][0]) / 2)
        # get all points inside the slab x0 with unbounded y and z
        allpointsleft = numpy.array([])
        allpointsright = numpy.array([])
        idxmappingleft = numpy.array([])
        idxmappingright = numpy.array([])
        npointsleft = 0
        npointsright = 0
        for i in range(n):
```

```

        if vectors[i][0] >= x0 - closest and vectors[i][0] < x0:
            allpointsleft = numpy.append(allpointsleft, vectors[i])
            idxmappingleft = numpy.append(idxmappingleft, i)
            npointsleft += 1
        if vectors[i][0] >= x0 and vectors[i][0] <= x0 + closest: # points in the middle
            included in the right, if there is (should be none)
            allpointsright = numpy.append(allpointsright, vectors[i])
            idxmappingright = numpy.append(idxmappingright, i)
            npointsright += 1
        allpointsleft = numpy.reshape(allpointsleft, (npointsleft, vectors[0].size))
        allpointsright = numpy.reshape(allpointsright, (npointsright, vectors[0].size))
        # print(allpointsleft, allpointsright)
        # print(idxmapping)
        # only need to consider these points in the slab

        # get shortest distance, compare all the points with y distance <= delta (closest) and z
        distance <= delta (closest) and so on for n-dimensions
        # there will always be a hard limit on the number of points for every point inside the
        slab, for 2D, it is 6, for 3D it is 18 and so on
        # thus  $O(n * n)$  will be  $O(n * k)$  where k is a constant,
        # therefore  $O(n)$ , even considering the pessimistic scenario where all points are in the
        slab

        nrowleft, ncolleft = allpointsleft.shape
        nrowright, ncolright = allpointsright.shape
        for i in range(0, nrowleft):
            for j in range(0, nrowright):
                p1 = allpointsleft[i]
                p2 = allpointsright[j]
                # get all distance of p1 and p2 in the y and z dimension (in n-dimension, get
                all n-1 dimension's distance)
                p3 = abs(p1 - p2)[1:]
                # check if their distances <= delta (closest)
                # this is the constraint (sparsity condition)
                if all(x <= closest for x in p3):
                    distance = getDistanceBetweenTwoPoints(p1, p2)
                    if distance < closest:
                        closest = distance
                        idxpair = numpy.array([idxmappingleft[i],
                        idxmappingright[j]]).astype(int)
                        # print(idxpair, idxpair[0])

            # Calculating the T(n) :  $O(n \log n)$  (from the DnC algorithm) +  $O(nk)$  (see the above comments
            for description) =  $O(n \log n)$ 
            return idxpair, numpy.round(closest, 3)

```

InputHandler.py

```

# Importing Libraries
from Modules.Sorting import partitioning
from Modules.Sorting import quickSort

import numpy
from numpy.random import Generator, PCG64
import time

class InputHandler():
    def __init__(self, num: int, dimension: int, size: int, decimal0n=True, randomize=True,
    inputVector=numpy.array([])):
        self.num = num
        self.dimension = dimension
        self.size = size
        self.vecArr = []

        # option to randomize the vectors or not
        if (randomize):
            self.randomizeVector()
        else:
            self.vecArr = inputVector
        # option to use decimals or not
        if not decimal0n:
            self.vecArr = numpy.round(self.vecArr)
        else:
            self.vecArr = numpy.round(self.vecArr, 3)

```

```

# Preprocessing
# Sorting Array
self.vecArr = numpy.transpose(self.vecArr)
quickSort(self.vecArr, self.dimension, 0, self.num-1)
self.vecArr = numpy.transpose(self.vecArr)

def randomizeVector(self):
    print("Randomizing vectors...")

    vector = []
    for i in range(self.num * self.dimension):
        a = numpy.random.uniform(low=0.0, high=self.size)
        vector.append(a)
    self.vecArr = numpy.array(vector).reshape(self.num, self.dimension)

    # self.vecArr = self.size * self.vecArr

def printVectors(self):
    print("Printing vectors...")
    print(self.vecArr)

```

Sorting.py

```

import numpy

# DIVIDE AND CONQUER
# QUICK SORT

def partitioning(vectors: numpy.array, dimension: int, lowerBound: int, upperBound: int):
    # Choose the pivoting
    pivot = vectors[0][upperBound]
    # Pointer
    i = lowerBound - 1

    # Traverse and comparing value with pivot
    for j in range(lowerBound, upperBound):
        # Swapping if value is lower
        if vectors[0][j] <= pivot:
            i += 1
            for k in range(dimension):
                (vectors[k][i], vectors[k][j]) = (vectors[k][j], vectors[k][i])
    # Swapping pivot element
    for p in range(dimension):
        (vectors[p][i+1], vectors[p][upperBound]) = (vectors[p][upperBound], vectors[p][i+1])
    # Return pointer
    return (i+1)

def quickSort(vectors: numpy.array, dimension: int, lowerBound: int, upperBound: int):
    if lowerBound < upperBound:
        # Finding pivot element
        x = partitioning(vectors, dimension, lowerBound, upperBound)
        # Recursive on the left and right of pivot
        quickSort(vectors, dimension, lowerBound, x-1)
        quickSort(vectors, dimension, x+1, upperBound)

```

Visualizer.py

```

import matplotlib.pyplot as plt
import numpy
from mpl_toolkits import mplot3d

def visualize2DResult(vectors: numpy.array, idxpair):
    print("Visualizing Vectors...")
    # Configure Plot
    xres = []
    yres = []
    xdata = []
    ydata = []

    # Set labels
    # plt.set_xlabel('X Label')
    # plt.set_ylabel('Y Label')

```

```

# Data for three-dimensional scattered points
for i in range(vectors.shape[0]):
    if i in idxpair:
        # data for results
        xres = numpy.append(xres, vectors[i,0])
        yres = numpy.append(yres, vectors[i,1])
    else:
        # data
        xdata = numpy.append(xdata, vectors[i,0])
        ydata = numpy.append(ydata, vectors[i,1])

plt.scatter(xdata, ydata, color= "black", linewidth=0.5)
plt.scatter(xres, yres, color="green", linewidth=0.5)
plt.savefig('Visualizer.png', dpi=300)

plt.show()

def visualize3DResult(vectors: numpy.array, idxpair):
    print("Visualizing Vectors...")
    # Configure Plot
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    count = 0
    RGB = [0,0,0]
    xres = []
    yres = []
    zres = []
    xdata = []
    ydata = []
    zdata = []
    x = []
    y = []
    z = []

    # Set labels
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    # Data for three-dimensional scattered points
    for i in range(vectors.shape[0]):
        if i in idxpair:
            # data for results
            xres = numpy.append(xres, vectors[i,0])
            yres = numpy.append(yres, vectors[i,1])
            zres = numpy.append(zres, vectors[i,2])
            count += 1
            if (count % 2 == 1):
                RGB[count % 3] = numpy.random.uniform(0.3, 1.0)
                x = []
                y = []
                z = []
                x = numpy.append(x, vectors[i,0])
                y = numpy.append(y, vectors[i,1])
                z = numpy.append(z, vectors[i,2])
            else:
                x = numpy.append(x, vectors[i,0])
                y = numpy.append(y, vectors[i,1])
                z = numpy.append(z, vectors[i,2])
                ax.plot(x, y, z, color='green')
        else:
            # data
            xdata = numpy.append(xdata, vectors[i,0])
            ydata = numpy.append(ydata, vectors[i,1])
            zdata = numpy.append(zdata, vectors[i,2])

    ax.scatter(xdata, ydata, zdata, color= "black", linewidth=0.5)
    ax.scatter(xres, yres, zres, color=RGB, linewidth=0.5)
    plt.savefig('../test/Visualizer.png', dpi=300)
    plt.show()

```

## BAB IV

### EKSPERIMEN DAN ANALISIS

Setelah dibuat algoritma penyelesaian menggunakan *Divide and Conquer*, hasilnya akan dibandingkan dengan algoritma *brute force* sebagai referensi. Parameter yang dijadikan perbandingan adalah kesamaan hasil penyelesaian, banyak operasi yang dilakukan, serta waktu yang diperlukan untuk menemukan penyelesaian.

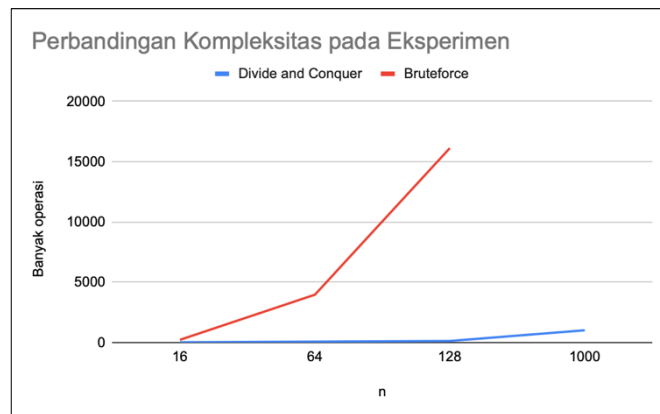
#### 4.1. Eksperimen Program

Eksperimen Program dilakukan pada perangkat dengan spesifikasi Prosesor Ryzen 5 3550H dan Intel i5.

Soal	Penyelesaian	Referensi / Brute Force	Analisis
N titik: 1000 N dimensi: 3 Batas koordinat: 1000	<b>START CONQUERING!</b> closest distance: 5.969 pair of points index: [769 773] 769: [782.26 113.743 529.057] 773: [785.975 109.526 531.068] number of operations: 1000 time duration: 0.20724 seconds	closest distance: 5.969 pair of points index: [769 773] 769: [782.26 113.743 529.057] 773: [785.975 109.526 531.068] number of operations: 998001 time duration: 20.01956 seconds Oof! That took a lot longer	Kedua algoritma mengembalikan hasil yang sama. Hal yang menarik muncul dari algoritma DnC, dimana hanya membutuhkan 1000 operasi untuk 100 titik
N titik: 16 N dimensi: 3 Batas koordinat: 100	<b>START CONQUERING!</b> closest distance: 19.286 pair of points index: [10 12] 10: [68.733 1.534 14.874] 12: [85.209 2.62 24.839] number of operations: 19 time duration: 0.00184 seconds	closest distance: 19.286 pair of points index: [10 12] 10: [68.733 1.534 14.874] 12: [85.209 2.62 24.839] number of operations: 225 time duration: 0.00188 seconds Oof! That took a lot longer	Kedua algoritma mengembalikan hasil yang sama, untuk n yang cukup kecil, perbedaan algoritma <i>divide and conquer</i> dan <i>brute force</i> belum cukup terlihat dari segi waktu, walau sudah terlihat dalam segi operasi, hal ini dikarenakan <i>overhead</i> dari pemanggilan rekursi masih menyeimbangkan waktu dari perhitungan operasi pada <i>brute force</i>
N titik: 64 N dimensi: 3 Batas koordinat: 100	<b>START CONQUERING!</b> closest distance: 5.591 pair of points index: [38 40] 38: [51.831 64.985 15.781] 40: [55.588 60.87 16.243] number of operations: 70 time duration: 0.00817 seconds	closest distance: 5.591 pair of points index: [38 40] 38: [51.831 64.985 15.781] 40: [55.588 60.87 16.243] number of operations: 3969 time duration: 0.03383 seconds Oof! That took a lot longer	Kedua algoritma menghasilkan hasil yang sama, dan mulai terlihat efek dari $O(n^2)$ dibandingkan hanya <i>overhead</i> pemanggilan rekursi
N titik: 128 N dimensi: 3 Batas koordinat: 100	<b>START CONQUERING!</b> closest distance: 2.245 pair of points index: [82 84] 82: [61.733 37.588 64.607] 84: [62.225 35.398 64.639] number of operations: 128 time duration: 0.01533 seconds	closest distance: 2.245 pair of points index: [82 84] 82: [61.733 37.588 64.607] 84: [62.225 35.398 64.639] number of operations: 16129 time duration: 0.12453 seconds Oof! That took a lot longer	Kedua algoritma menghasilkan hasil yang sama, dapat dilihat bahwa jarak antar operasi dan durasi memiliki jarak yang semakin menjauh

N titik: 1000 N dimensi: 3 Batas koordinat: 100	<b>START CONQUERING!</b> closest distance: 0.674 pair of points index: [897 902] 897: [89.848 20.589 46.831] 902: [90.176 20.349 47.369] number of operations: 1015 time duration: 0.17993 seconds	closest distance: 0.674 pair of points index: [897 902] 897: [89.848 20.589 46.831] 902: [90.176 20.349 47.369] number of operations: 998001 time duration: 7.29592 seconds Oof! That took a lot longer	Kedua algoritma menghasilkan hasil yang sama dengan perbedaan jumlah operasi yang sangat signifikan
--	--	---	---

## 4.2. Perbandingan Kompleksitas Algoritma dengan Brute Force



Gambar 4.2.1. Perbandingan Kompleksitas Algoritma pada Eksperimen

Berikut merupakan perbandingan kompleksitas algoritma berdasarkan eksperimen yang dilakukan. Pada  $n = 1000$ , banyak operasi yang dilakukan algoritma *bruteforce* adalah sebanyak 998001 operasi sehingga jika di *plot*, dikhawatirkan tidak merepresentasikan kompleksitas algoritma *divide and Conquer*.

Berdasarkan eksperimen tersebut, didapat bahwa kompleksitas algoritma *brute force* dalam pemecahan masalah pencarian pasangan titik terdekat memerlukan  $O(n^2)$  karena untuk setiap titik, diperlukan untuk mengecek jarak ke setiap titik lainnya. Sedangkan untuk algoritma *divide and conquer*, karena untuk setiap himpunan akan dibagi menjadi 2 himpunan dengan jumlah titik masing-masing setengah dari total titik di himpunan awal, maka

$$T(n) = 2 T\left(\frac{n}{2}\right) + n(c^{d-1}), n > 2$$

Dimana  $n(c^{d-1})$ ,  $c$  sebuah konstan dan  $d$  dimensi persoalan, adalah waktu untuk memroses titik-titik didekat garis pembagi, dan

$$T(n) = a, 2 \leq n \leq 3$$

Sehingga dengan teorema master,  $T(n) = O(n \log n)$

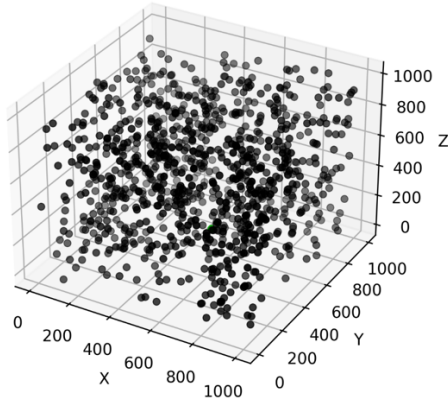
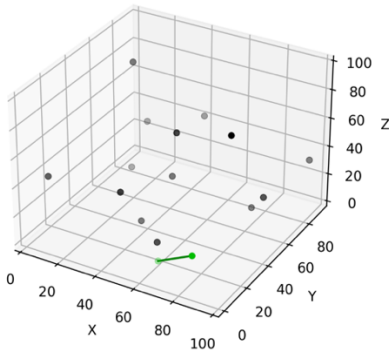
## 4.3. Algoritma Pengurutan (*QuickSort*)

Dalam tugas kecil ini, kami memilih pendekatan *divide and conquer* dalam algoritma pengurutan. Langkah yang dilakukan pada algoritma yaitu mempartisi larik menjadi lebih kecil. Pada implementasi yang penulis lakukan, *pivot* untuk mempartisi selalu memilih elemen terakhir. Ketika elemen dibandingkan dan terdapat ketidaksesuaian, maka akan terjadi pergantian elemen.



### 4.3. Masukan dan Keluaran Program

Berikut merupakan masukan dan keluaran program. Program menerima 3 parameter, yaitu jumlah titik (n), dimensi (d), dan threshold (t).

Masukan	Hasil
n : 1000 titik d : 3 t : 1000	 <p>Both methods returns the same results!</p> <p>Closest Distance : 5.969</p> <p>Pair of points index : [769 773]</p> <p>769: [782.26 113.743 529.057]</p> <p>773: [785.975 109.526 531.068]</p> <p>DNC Statistics : 1000 operations at 0.20724 seconds</p> <p>BF Statistics : 998001 operations at 20.01956 seconds</p>
n : 16 titik d : 3 t : 100	 <p>Both methods returns the same results!</p> <p>Closest Distance : 19.286</p> <p>Pair of points index : [10 12]</p> <p>10: [68.733 1.534 14.874]</p> <p>12: [85.209 2.62 24.839]</p> <p>DNC Statistics : 19 operations at 0.00184 seconds</p> <p>BF Statistics : 225 operations at 0.00188 seconds</p>

<p>n : 64 titik d : 3 t : 100</p>	<div data-bbox="636 210 1052 577" data-label="Figure"> </div> <p>Both methods returns the same results!</p> <p>Closest Distance : 5.591</p> <p>Pair of points index : [38 40] 38: [51.831 64.985 15.781] 40: [55.588 60.87 16.243]</p> <p>DNC Statistics : 70 operations at 0.00817 seconds</p> <p>BF Statistics : 3969 operations at 0.03383 seconds</p>
<p>n : 128 titik d : 3 t : 100</p>	<div data-bbox="571 882 836 1123" data-label="Figure"> </div> <div data-bbox="961 871 1242 1123" data-label="Figure"> </div> <p>Both methods returns the same results!</p> <p>Closest Distance : 2.245</p> <p>Pair of points index : [82 84] 82: [61.733 37.588 64.607] 84: [62.225 35.398 64.639]</p> <p>DNC Statistics : 128 operations at 0.01533 seconds</p> <p>BF Statistics : 16129 operations at 0.12453 seconds</p>
<p>n : 1000 titik d : 3 t : 100</p>	<div data-bbox="571 1407 812 1627" data-label="Figure"> </div> <div data-bbox="977 1396 1258 1627" data-label="Figure"> </div> <p>Both methods returns the same results!</p> <p>Closest Distance : 0.674</p> <p>Pair of points index : [897 902] 897: [89.848 20.589 46.831] 902: [90.176 20.349 47.369]</p> <p>DNC Statistics : 1015 operations at 0.17993 seconds</p> <p>BF Statistics : 998001 operations at 7.29592 seconds</p>

Berikut merupakan permasalahan dalam Dimensi-n.

Permasalahan	Hasil
n : 1000 titik d : 5 t : 1000	<pre> Randomize vectors? (y/n) : y Enter The Number of Points : 1000 Enter The Number of Dimensions : 5 Enter max threshold size for a point : 1000 Use decimals? (y/n) : y Randomizing vectors...  Divide and Conquer START CONQUERING! closest distance: 32.441 pair of points index: [231 244] 231: [241.743 256.396 89.701 218.531 985.103] 244: [250.881 268.71 88.717 191.215 993.48 ] number of operations: 1374 time duration: 0.2436 seconds  Bruteforce closest distance: 32.441 pair of points index: [231 244] 231: [241.743 256.396 89.701 218.531 985.103] 244: [250.881 268.71 88.717 191.215 993.48 ] number of operations: 998001 time duration: 9.69494 seconds </pre>
Both methods returns the same results! Closest Distance : 32.441 Pair of points index : [231 244] 231: [241.743 256.396 89.701 218.531 985.103] 244: [250.881 268.71 88.717 191.215 993.48 ] DNC Statistics : 1374 operations at 0.2436 seconds BF Statistics : 998001 operations at 9.69494 seconds	
n : 1000 titik d : 10 t : 1000	<pre> Randomize vectors? (y/n) : y Enter The Number of Points : 1000 Enter The Number of Dimensions : 10 Enter max threshold size for a point : 1000 Use decimals? (y/n) : y  Divide and Conquer START CONQUERING! closest distance: 279.359 pair of points index: [722 910] 722: [721.13 447.046 108.994 813.956 696.734 782.862 672.114 369.759 279.138 697.996] 910: [918.104 541.815 88.694 883.09 778.365 821.989 634.324 294.797 214.768 773.39 ] number of operations: 5348 time duration: 1.44834 seconds  Bruteforce closest distance: 279.359 pair of points index: [722 910] 722: [721.13 447.046 108.994 813.956 696.734 782.862 672.114 369.759 279.138 697.996] 910: [918.104 541.815 88.694 883.09 778.365 821.989 634.324 294.797 214.768 773.39 ] number of operations: 998001 time duration: 59.83016 seconds Oof! That took a lot longer </pre>

Both methods returns the same results!

Closest Distance : 279.359

Pair of points index : [722 910]

722: [721.13 447.046 108.994 813.956 696.734 782.862 672.114 369.759 279.138 697.996]

910: [918.104 541.815 88.694 883.09 778.365 821.989 634.324 294.797 214.768 773.39 ]

DNC Statistics : 5348 operations at 1.44834 seconds

BF Statistics : 998001 operations at 59.83016 seconds

n : 1000 titik

d : 1000

t : 1000

```
Randomize vectors? (y/n) : y
Enter The Number of Points : 1000
Enter The Number of Dimensions : 1000
Enter max threshold size for a point : 1000
Use decimals? (y/n) : y
Randomizing vectors...
START CONQUERING!
closest distance: 11704.082
pair of points index: [101 727]
101: [100.646 630.876 318.456 927.561 107.867 486.853 566.671 345.089 22.62
660.738 710.584 189.98 605.43 209.795 15.804 407.552 266.498 171.548
926.057 465.558 166.742 301.985 8.739 71.5 356.121 862.171 232.65
130.28 680.172 591.886 159.517 263.779 109.08 871.859 959.448 808.499
278.862 879.374 542.025 504.854 258.911 191.331 505.143 589.843 799.76
66.455 652.191 3.27 246.486 407.687 456.173 593.393 815.026 835.318
492.519 36.07 648.338 761.689 285.637 928.249 639.26 258.809 983.073
170.576 947.106 380.957 1.842 890.984 469.822 913.949 98.708 213.831
486.316 648.743 733.882 692.497 898.608 134.634 668.983 893.572 165.371
117.028 364.906 187.128 873.991 858.36 161.909 261.758 778.645 892.096
865.367 820.091 587.656 428.116 140.357 250.21 223.121 288.787 860.491
373.817 577.116 825.649 769.888 234.004 506.116 456.374 439.159 436.264
220.5 660.134 70.919 614.242 989.071 326.388 329.509 967.526 95.159
557.984 814.721 504.643 664.608 492.856 978.963 696.467 554.257 890.705
663.824 903.501 287.969 250.618 866.887 847.949 32.938 246.149 538.855
526.814 602.836 786.182 768.285 69.01 908.194 713.051 185.36 683.446
...
5.59232e+02 7.22422e+02 8.07750e+02 7.11605e+02 4.37923e+02 2.27704e+02
7.56769e+02 8.80496e+02 2.07675e+02 2.30839e+02 2.66255e+02 9.61395e+02
9.17016e+02 9.52000e-01 2.82895e+02 3.73516e+02 5.19436e+02 8.94395e+02
1.94704e+02 9.17897e+02 4.79416e+02 7.71975e+02 6.02556e+02 5.80769e+02
4.81334e+02 2.99083e+02 2.23391e+02 9.22715e+02 3.22942e+02 3.31790e+02
1.91317e+02 7.50530e+02 2.72935e+02 9.95160e+01 9.38934e+02 8.22177e+02
1.01040e+01 7.07895e+02 6.01203e+02 5.23620e+01 9.24500e+00 7.15288e+02
6.10502e+02 1.84917e+02 8.22595e+02 2.67812e+02 6.66691e+02 9.16013e+02
5.27490e+01 8.56115e+02 6.99878e+02 2.78812e+02 6.44642e+02 1.86482e+02
7.73006e+02 7.90816e+02 4.18779e+02 5.86450e+02 5.54770e+02 8.03531e+02
3.32883e+02 9.21879e+02 8.34759e+02 6.16170e+02 6.73375e+02 4.58234e+02
8.05790e+02 4.25097e+02 4.17930e+02 1.19783e+02 5.39125e+02 6.09155e+02
8.19863e+02 1.88447e+02 4.99801e+02 4.41010e+02 6.15310e+02 6.63307e+02
6.09568e+02 6.90036e+02 7.30890e+02 1.96381e+02 9.03098e+02 4.81578e+02
1.61760e+01 8.06312e+02 4.24013e+02 2.55616e+02 6.85683e+02 7.90267e+02
8.52011e+02 2.97553e+02 8.97159e+02 2.91200e+01 7.43942e+02 2.43123e+02
6.30995e+02 5.07898e+02 5.38051e+02 2.96710e+02 2.57207e+02 3.58086e+02
3.18274e+02 3.38868e+02 4.66946e+02 2.24546e+02 5.17656e+02 2.05963e+02
7.16067e+02 9.94690e+02 6.99488e+02 1.84574e+02 6.86472e+02 3.78091e+02
3.96666e+02 4.99008e+02 4.10019e+02 7.68691e+02 7.59977e+02 3.39857e+02
3.15710e+02 3.83679e+02 6.21125e+02 6.53146e+02 7.96592e+02 6.20870e+02
5.46298e+02 2.49334e+02 5.22576e+02 3.04469e+02 6.10318e+02 4.05886e+02
5.45243e+02 4.82867e+02 9.48744e+02 5.48022e+02 7.91453e+02 3.24074e+02
8.45500e+01 3.80008e+02 9.69441e+02 6.38398e+02 9.04442e+02 6.79618e+02
1.63027e+02 4.80485e+02 5.99279e+02 6.56571e+02 3.87744e+02 4.61358e+02
6.66167e+02 4.44099e+02 8.94681e+02 8.83020e+01 6.63369e+02 1.63450e+02
9.16191e+02 8.25716e+02 7.28845e+02 1.45300e+01 3.74730e+02 3.85724e+02
5.70232e+02 5.61668e+02 2.35435e+02 9.01226e+02]
number of operations: 499500
time duration: 67.76996 seconds
```

Diketahui dapat melakukan operasi dalam 67 detik sebanyak 499500 operasi. Bruteforce membutuhkan lebih dari 3 jam sehingga tidak terselesaikan.

Berikut merupakan hasil dari permasalahan menarik yang penulis temukan selama mengerjakan tugas kecil ini.

Permasalahan Menarik	Hasil
100.000 titik 3 dimensi Batas: 1	<pre> Randomize vectors? (y/n) : y Enter The Number of Points : 100000 Enter The Number of Dimensions : 3 Enter max threshold size for a point : 1 Use decimals? (y/n) : y Randomizing vectors... START CONQUERING! closest distance: 0.001 pair of points index: [ 310 402 2548 2675 4918 5006 5604 5646 5877 6026 9097 9209 12041 12101 13607 13696 14161 14278 16089 16170 18217 18313 20824 20867 20905 20983 25163 25224 31743 31806 32871 32966 33314 33395 34070 34217 34508 34626 43437 43546 44899 44987 47752 47804 50434 50615 55749 55793 56712 56785 68385 68526 69788 69871 74542 74673 75215 75273 81936 81986 84124 84233 87841 87866 88351 88465 91492 91628 93215 93341 96575 96695] 310: [0.003 0.751 0.462] 402: [0.004 0.751 0.462] number of operations: 102802 time duration: 90.00496 seconds NOW THE PROBLEM HAS BEEN CONQUERED! DEVIDE ET IMPERA! </pre> <p>Pada eksperimen ini, ditemukan bahwa titik terdekat dalam 1 juta titik dengan threshold 0-1000 dapat diperoleh selama 90.004 detik sebanyak 102802 operasi. Bruteforce melebihi 3 jam sehingga tidak terselesaikan.</p>
1.000.000 titik 3 dimensi Batas: 1000	<pre> closest distance: 0.111 pair of points index: [457180 457257] 457180: [457.169 819.062 267.144] 457257: [457.247 819.095 267.22 ] number of operations: 2024287 time duration: 12464.03732 seconds NOW THE PROBLEM HAS BEEN CONQUERED! DEVIDE ET IMPERA!  Now Bruteforcing your way through.... Traceback (most recent call last):   File "C:\Users\Kenneth Ezekiel\OneDrive\Documents\GitHub\Tucil2_13521089_13521171\src\main.py", line 96, in &lt;module&gt;     pairBF, distBF = Bf.bruteforce(Input.vecArr)   File "C:\Users\Kenneth Ezekiel\OneDrive\Documents\GitHub\Tucil2_13521089_13521171\src\Modules\Bruteforce.py", line 24, in bruteforce     temp += pow(vecBar[p],2) KeyboardInterrupt </pre> <p>Pada eksperimen ini, ditemukan bahwa titik terdekat dalam 1 juta titik dengan threshold 0-1000 dapat diperoleh selama 12464.037 detik sebanyak 2024287 operasi. Bruteforce melebihi 7 jam sehingga tidak terselesaikan.</p>
1.000.000 titik 3 dimensi Batas: 10000	<pre> Randomize vectors? (y/n) : y Enter The Number of Points : 1000000 Enter The Number of Dimensions : 3 Enter max threshold size for a point : 10000 Use decimals? (y/n) : y Randomizing vectors... START CONQUERING! closest distance: 0.956 pair of points index: [241439 241465] 241439: [2425.463 421.743 7113.994] 241465: [2425.735 421.98 7113.109] number of operations: 1058045 time duration: 2556.59263 seconds NOW THE PROBLEM HAS BEEN CONQUERED! DEVIDE ET IMPERA! </pre> <p>Pada eksperimen ini, ditemukan bahwa titik terdekat dalam 1 juta titik dengan threshold 0-10000 dapat diperoleh selama 2556.592 detik sebanyak 1058045 operasi. Bruteforce melebihi 7 jam sehingga tidak terselesaikan</p>

## BAB V

### KESIMPULAN DAN SARAN

#### 5.1. Kesimpulan

Pada tugas kecil II IF2211 Strategi Algoritma ini telah diimplementasikan algoritma berbasis *Divide and Conquer* beserta fungsi-fungsi pendukung dalam tujuan untuk menyelesaikan pencarian pasangan titik terdekat pada ruang  $n$  dimensi. Fungsi-fungsi tersebut mencakup fungsi yang menangani masukan titik, fungsi penyelesaian, fungsi pengurutan, serta fungsi visualisasi. Implementasi tersebut kemudian berhasil direalisasikan dalam sebuah program dengan bahasa python.

Algoritma *divide and conquer* pada tugas ini digunakan untuk penyelesaian masalah pencarian pasangan titik terdekat, dimana himpunan titik akan dibagi menjadi bagian yang semakin lama semakin kecil, dihitung jarak terkecil dari tiap himpunan kecil alias *local minimum*, dan menggabungkannya untuk mencari jarak terkecil diantara pasangan titik di himpunan titik awal yang bisa disebut *global minimum*.

Dengan demikian, penulis menyimpulkan bahwa melalui Tugas Kecil II IF2211 Strategi Algoritma ini, dapat dibuat sebuah algoritma berbasis *Divide and Conquer* yang mengkomputasi solusi permasalahan *closest pair* pada titik-titik di dalam ruang dimensi  $N$ .

#### 5.2. Saran

Tugas Kecil IF2211 Strategi Algoritma Semester II Tahun 2022/2023 menjadi salah satu tugas yang memberikan pelajaran baru bagi penulis. Berdasarkan pengalaman penulis mengerjakan tugas ini, berikut merupakan saran untuk pembaca yang ingin melakukan atau mengerjakan hal yang serupa.

1. Keefektifan dalam kerja sama tim merupakan hal yang penting dalam mengerjakan tugas ini. Tugas ini sangat terbantu oleh pemakaian *real-time collaboration app*. Selain itu, pemakaian aplikasi pengelola version control seperti Github sangat disarankan agar memudahkan untuk mengelola pekerjaan secara asinkron.
2. Mengingat bahwa tugas ini berhubungan erat dengan perhitungan secara matematis, dibutuhkan pemahaman serta ketelitian yang ekstra dalam pengembangan algoritma. Setiap strategi yang diimplementasi perlu diperhatikan kebenarannya dan meminimalisasi terjadi berkurangnya informasi dalam pengolahan data.
3. Mengingat bahwa tugas ini menangani titik dalam  $n$  dimensi, perancangan algoritma dan struktur program menjadi hal yang penting untuk diperhatikan. Pada tugas ini, kami menggunakan tipe *multidimensional array* yang banyak diolah menggunakan Pustaka *numpy*.

## DAFTAR PUSTAKA

Gautam, S. (n.d.). *Divide and*

*Conquer Algorithm*. <https://www.enjoyalgorithms.com/blog/divide-and-conquer>

*Python File Write*. (n.d.). [https://www.w3schools.com/python/python\\_file\\_write.asp](https://www.w3schools.com/python/python_file_write.asp)

*Closest pair of points in 3D*. (n.d.). Computer Science Stack

Exchange. <https://cs.stackexchange.com/questions/155286/closest-pair-of-points-in-3d>

Suri, S. (n.d.). UC Santa Barbara : Closest Pair Problem.

<https://sites.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf>

Munir, R. (2022). Algoritma Divide and Conquer (1). Retrieved from Homepage

Rinaldi Munir: [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)

Munir, R. (2022). Algoritma Divide and Conquer (2). Retrieved from Homepage

Rinaldi Munir: [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, R. (2022). Algoritma Divide and Conquer (3). Retrieved from Homepage

Rinaldi Munir: [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)

Munir, R. (2022). Algoritma Divide and Conquer (4). Retrieved from Homepage

Rinaldi Munir: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian4.pdf)

## LAMPIRAN

### Pranala Repositori Github

[Repositori GitHub](#)

### Checklist Fitur

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