TUGAS KECIL 2 IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2022/2023

MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA DIVIDE AND CONQUER

Disusun Oleh:

13521059 Arleen Chrysantha Gunardi

13521143 Raynard Tanadi

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

DAFTAR GAMBAR

BAB I
DESKRIPSI MASALAH

1.1 Deskripsi Masalah

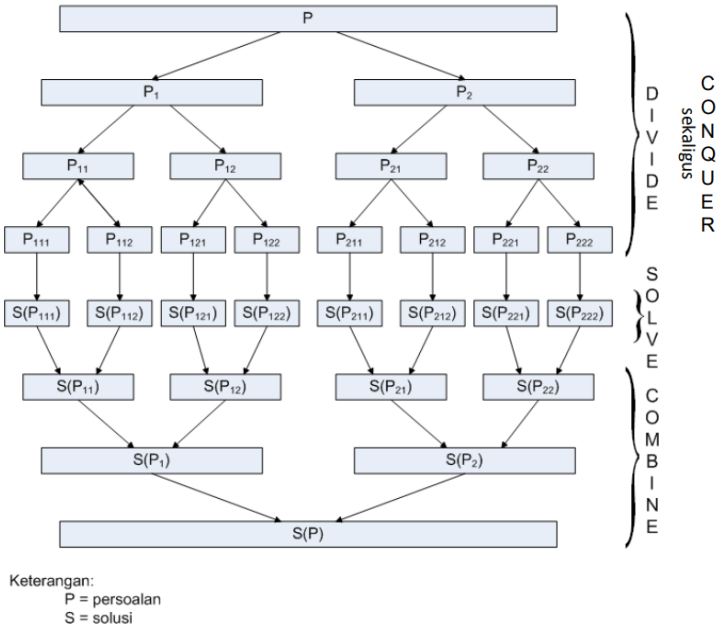
Misalkan terdapat sekumpulan titik-titik pada bidang tiga dimensi. Di antara kumpulan titik tersebut, terdapat paling sedikit dua titik yang memiliki jarak terdekat dibandingkan dengan titik-titik lainnya. Pasangan titik terdekat tersebut dapat dicari untuk berbagai macam kepentingan dalam dunia nyata, misalnya untuk mencari dua gambar wajah yang paling mirip pada *face recognition*, mencari dua pesawat terdekat dalam sistem lalu-lintas udara untuk menghindari kecelakaan, dan lain-lain.

Mencari pasangan titik terdekat di antara sekumpulan titik dapat dilakukan dengan mengimplementasikan strategi algoritma Divide and Conquer. Pada bidang tiga dimensi, titik direpresentasikan sebagai koordinat (x, y, z). Dengan memanfaatkan perhitungan jarak Euclidean, akan didapatkan pasangan titik terdekat $P1(x_1, y_1, z_1)$ dan $P2(x_1, y_1, z_1)$ di antara n titik pada bidang tiga dimensi.

BAB II
LANDASAN TEORI

2.1 Algoritma Divide and Conquer

Algoritma Divide and Conquer merupakan salah satu strategi algoritma untuk memecahkan berbagai macam persoalan. Terdapat tiga langkah utama pada algoritma ini, yaitu *divide*, *conquer*, dan *combine*. Pertama, persoalan dibagi (*divided*) menjadi beberapa upa-persoalan yang memiliki karakteristik yang mirip dengan persoalan utuhnya. Lalu, setiap upa-persoalan dipecahkan (*conquered*) secara terpisah. Terakhir, solusi dari setiap upa-persoalan digabungkan (*combined*) sehingga menghasilkan solusi dari persoalan yang utuh.



Gambar 2.1 Ilustrasi Algoritma Divide and Conquer

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf

Adapun implementasi dari algoritma Divide and Conquer dapat digunakan untuk memecahkan persoalan-persoalan seperti pencarian nilai ekstrem, proses pengurutan (*sorting*) larik, perhitungan perpangkatan, atau pencarian pasangan titik terdekat pada sekumpulan titik pada bidang tertentu. Dengan strategi Divide and Conquer, persoalan dapat dipecahkan secara lebih efektif jika dibandingkan dengan strategi Brute Force.

2.2 Algoritma Brute Force

Algoritma Brute Force adalah algoritma yang menggunakan pendekatan yang lempang atau *straightforward* untuk memecahkan suatu persoalan. Algoritma ini juga memiliki beberapa karakteristik, yaitu sederhana, langsung, dan jelas caranya, sehingga dalam algoritmanya, dibutuhkan jumlah langkah yang besar dalam penyelesaiannya. Dengan metode *exhaustive search*, algoritma ini mencari dan mengecek semua kemungkinan kejadian. Akibatnya, algoritma ini memiliki kompleksitas waktu sebesar $O(nm)$.

2.3 Quicksort

Pencarian pasangan titik terdekat di antara sekumpulan titik pada ruang n dimensi dapat dilakukan dengan strategi algoritma Divide and Conquer. Namun, kumpulan titik yang diproses merupakan kumpulan titik yang sudah terurut membesar berdasarkan posisinya. Oleh karena itu, dibutuhkan suatu proses pengurutan (*sorting*).

Salah satu algoritma *sorting* yang memanfaatkan strategi Divide and Conquer adalah Quicksort. Prinsip dari algoritma Quicksort adalah dengan membagi larik berukuran n menjadi dua upalarik berukuran $\frac{n}{2}$. Kemudian, masing-masing upalarik diurutkan dan digabungkan sehingga larik menjadi terurut. Adapun garis besar algoritma Quicksort untuk mengurutkan larik terurut membesar adalah sebagai berikut.

1. Pilih sebuah elemen x dari larik A sebagai pivot.
2. Pindai "upalarik" bagian kiri – yaitu upalarik yang berada di sebelah kiri pivot – sehingga ditemukan sebuah elemen pertama yang lebih besar daripada pivot ($A[p] \geq x$).
3. Pindai "upalarik" bagian kanan – yaitu upalarik yang berada di sebelah kanan pivot – sehingga ditemukan sebuah elemen pertama yang lebih kecil daripada pivot ($A[q] \leq x$).
4. Tukar posisi $A[p]$ dengan $A[q]$ pada larik.
5. Ulangi langkah (3) dengan posisi $p + 1$ dan $q - 1$. Pengulangan dilakukan hingga mencapai kondisi ketika $p \geq q$.

Dengan demikian, larik dapat diurutkan dengan algoritma Quicksort dengan kompleksitas waktu sebesar $O(n \log_2 n)$ untuk kasus terbaik dan kasus rata-rata atau $O(n^2)$ untuk kasus terburuk.

2.4 Pencarian Pasangan Titik Terdekat 3D

Pencarian pasangan titik terdekat merupakan persoalan ketika terdapat himpunan titik yang terdiri dari *n* buah titik pada bidang *d* dimensi, serta dilakukan pencarian sepasang titik pada himpunan tersebut yang memiliki jarak terdekat. Jarak tersebut dapat dicari dengan perhitungan Euclidean Distance.

Pada Tugas Kecil 2 IF2211 Strategi Algoritma ini, persoalan tersebut akan dilakukan pada bidang tiga dimensi dan/atau *n* dimensi. Persoalan ini dapat diselesaikan menggunakan setidaknya dua jenis algoritma berikut, yaitu Brute Force dan Divide and Conquer. Untuk menghitung jarak antara dua titik pada bidang tiga dimensi, perhitungan rumus Euclidean Distance adalah sebagai berikut:

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

BAB III
APLIKASI STRATEGI DIVIDE AND CONQUER

3.1 Aplikasi Algoritma Divide and Conquer pada Quicksort

Quicksort merupakan algoritma pengurutan secara Divide and Conquer dan termasuk ke dalam pendekatan sulit membagi, mudah menggabung. Pada Quicksort yang diimplementasikan, digunakan pendekatan penentuan elemen terakhir sebagai pivot. Kompleksitas kasus terbaik dari algoritma pengurutan ini adalah $O(n\ logn)$ dan $O(n^2)$ untuk kasus terburuk.

Pertama-tama, untuk melakukan pengurutan dengan algoritma Quicksort, dibutuhkan larik yang akan diurutkan, indeks awal, dan indeks akhir. Lalu, akan dilakukan pengecekan apakah indeks awal lebih kecil daripada indeks akhir untuk memastikan jika larik berisi lebih dari satu elemen. Jika hanya terdapat satu elemen pada larik, maka larik akan langsung dikembalikan.

Setelah itu, algoritma Quicksort akan memilih elemen terakhir dari larik sebagai pivot. Lalu, akan dilakukan pengecekan apakah terdapat elemen yang lebih kecil atau sama dengan dari pivot pada elemen pertama sampai elemen terakhir larik. Bila terdapat elemen yang lebih kecil atau sama dengan pivot, maka posisi elemen tersebut akan ditukar dengan elemen pada indeks ke-i untuk memajukan elemen ke depan. Pada akhir proses, larik akan terbagi menjadi dua buah upalarik yang sudah dikelompokkan secara kecil dan besar.

Terakhir, proses ini akan dilakukan secara rekursif untuk upalarik kiri dan upalarik kanan hingga setiap upalarik hanya memiliki satu elemen. Dengan demikian, larik menjadi terurut.

3.2 Aplikasi Algoritma Brute Force pada Pencarian Pasangan Titik Terdekat 3D

Pencarian pasangan titik terdekat dapat dicari salah satunya dengan menggunakan algoritma Brute Force. Kompleksitas dari algoritma ini adalah $O(n^2)$.

Pertama-tama, untuk melakukan pencarian pasangan titik terdekat dengan strategi Brute Force, dibutuhkan larik yang berisi himpunan titik pada bidang tiga dimensi. Larik tersebut minimal berisi dua buah titik, jumlah titik, dan dimensi (pada kasus kali ini dapat dilakukan pencarian terhadap titik pada bidang tiga dimensi maupun *n* dimensi). Lalu, program akan menyimpan nilai minimum yang didapatkan dari jarak titik pada indeks ke-0 dan ke-1. Setelah

itu, akan dicek apakah list berisi lebih dari atau sama dengan 3 titik. Jika tidak, maka program telah selesai dan mengembalikan nilai minimum tersebut.

Jika terdapat lebih dari atau sama dengan 3 titik, maka program akan melakukan iterasi bersarang i. Variabel i akan diiterasi dari indeks ke-0 sampai ke-$(n-1)$. Iterasi kedua akan mengiterasi dari index j. Variabel j akan diiterasi dari indeks ke-$(i+1)$ sampai ke-$(n-1)$. Untuk setiap iterasi j, jika i tidak sama dengan 0 atau j tidak sama dengan 0 (untuk menghindari pengecekan penetapan nilai minimum di awal yaitu i = 0 dan j = 1), maka akan dihitung jarak dari titik pada indeks ke-i dan indeks ke-j. Apabila jarak kedua titik tersebut lebih kecil dari nilai jarak minimum sementara, maka jarak tersebut akan menjadi nilai minimum yang baru.

Terakhir, jika iterasi sudah selesai, program akan mengembalikan pasangan titik terdekat di antara himpunan titik beserta jarak kedua titik tersebut.

3.3 Aplikasi Algoritma Divide and Conquer pada Pencarian Pasangan Titik Terdekat 3D

Untuk mencari pasangan titik terdekat pada sekumpulan titik pada bidang tiga dimensi (3D), diterapkan algoritma Divide and Conquer. Strategi ini menggunakan implementasi dari fungsi rekursif. Misalkan terdapat sekumpulan n titik pada bidang tiga dimensi P, maka basis rekursi adalah kondisi ketika $n \geq 2$ atau $n \leq 3$. Ketika hanya terdapat dua atau tiga buah titik pada P, maka jarak titik dapat dihitung langsung dengan rumus Euclidean (1) dan dibandingkan secara langsung sehingga didapatkan pasangan titik dengan jarak terdekat. Namun, ketika jumlah titik pada P lebih dari tiga, maka akan diterapkan strategi Divide and Conquer yang memanfaatkan fungsi rekursif.

Seperti yang telah dijelaskan pada bab sebelumnya, strategi Divide and Conquer terdiri dari tiga tahap, yaitu *divide*, *conquer*, dan *combine*. Pada tahap *divide*, larik titik P berukuran n yang sudah terurut membesar dibagi menjadi dua buah upalarik, masing-masing berukuran $\frac{n}{2}$. Misalkan P_{left} dan P_{right} adalah upalarik kiri dan kanan dari P, maka dengan pemanggilan fungsi rekursif, akan dicari pasangan titik terdekat dari masing-masing upalarik P_{left} dan P_{right}. Dengan demikian, didapatkan dua pasang titik yang merupakan pasangan titik terdekat pada upalarik P_{left} dan upalarik P_{right} (tahap *conquer*). Jarak kedua pasang titik tersebut dibandingkan sehingga pasangan titik dengan jarak terdekat yang dipilih (tahap *combine*). Dari tahap ini,

pasangan titik terdekat di antara upalarik P_{left} dan P_{right} didapatkan, yaitu misalkan titik a dan titik b. Akan tetapi, terdapat area yang perlu diperiksa juga, yaitu titik-titik pada area sekitar perbatasan yang membagi P_{left} dan P_{right}.

Untuk mencari pasangan titik terdekat pada area perbatasan, strategi Divide and Conquer kembali diterapkan. Misalkan P_{center} adalah upalarik titik pada area perbatasan, yaitu titik-titik yang memiliki selisih absis terhadap titik tengah yang kurang dari jarak terdekat antara titik a dan b – pasangan terdekat sementara dari gabungan upalarik kiri dan kanan (tahap *divide*). Dari semua titik yang ada pada P_{center}, akan diperiksa titik-titik mana yang memiliki jarak yang lebih dekat dibandingkan dengan jarak antara pasangan titik terdekat sementara a dan b. Metode untuk memeriksa jarak tersebut dengan efisien adalah dengan melakukan pengecekan terhadap jarak titik pada semua sumbu. Ketika jarak sepasang titik pada sebuah sumbu lebih besar daripada jarak terdekat sementara, maka pasangan titik tersebut dilewat. Hal ini dilakukan karena jarak antara pasangan titik tersebut tidak mungkin lebih dekat dibandingkan pasangan terdekat sementara jika jarak pasangan titik pada suatu sumbu lebih besar daripada jarak terdekat sementara. Sebaliknya, jika jarak pasangan titik pada semua sumbu lebih kecil dibandingkan jarak terdekat sementara, maka ada kemungkinan pasangan titik tersebut lebih dekat dibandingkan pasangan terdekat sementara. Lalu, jarak antara pasangan titik tersebut dikalkulasi dan dibandingkan dengan jarak terdekat sementara. Jika lebih dekat dibandingkan dengan jarak terdekat sementara, maka pasangan titik tersebut menjadi pasangan titik terdekat sementara yang baru. Hal tersebut dilakukan pada semua pasangan titik pada P_{center}. Dengan demikian, didapatkan pasangan titik terdekat di antara titik-titik pada area perbatasan, misalkan titik c dan titik d (tahap *conquer*).

Setelah proses tersebut, didapatkan dua pasangan titik, yaitu pasangan titik terdekat di antara P_{left} dan P_{right} (titik a dan titik b), serta pasangan titik terdekat pada P_{center} (titik c dan titik d). Jarak antara kedua pasangan titik tersebut dibandingkan (tahap *combine*). Pasangan titik dengan jarak terdekat dipilih sebagai pasangan titik terdekat di antara kumpulan titik P dan menjadi solusi dari larik himpunan titik secara keseluruhan.

Dengan algoritma di atas, dapat diketahui bahwa representasi dari efisiensi kompleksitas waktu algoritma ini dalam notasi Big-O adalah $O(n\ (log_2\ n)^2)$.

3.4 Aplikasi Algoritma Divide and Conquer pada Pencarian Pasangan Titik Terdekat pada Kumpulan Vektor R^n.

Untuk mencari pasangan titik terdekat pada sekumpulan vektor pada ruang orde n, tahapan algoritma yang dilakukan mirip dengan yang dilakukan pada bidang tiga dimensi, hanya saja jumlah pengulangan pengecekannya yang berbeda. Hal tersebut disebabkan karena dimensi yang perlu dicek berbeda. Misalnya, untuk bidang dua dimensi hanya perlu pengecekan untuk sumbu-x dan sumbu-y, sedangkan pengecekan pada bidang tiga dimensi dilakukan terhadap sumbu-x, sumbu-y, dan sumbu-z. Dengan demikian, dapat disimpulkan bahwa representasi dari efisiensi kompleksitas waktu algoritma untuk d dimensi dalam notasi Big-O adalah $O(n\,(log_2\,n)^{d-1})$.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

Program pencarian pasangan titik terdekat diimplementasikan dengan menggunakan bahasa pemrograman Python. Secara garis besar, program terdiri dari beberapa modul yang disimpan pada beberapa *file*, yaitu main.py untuk program utama, IO_processing.py untuk pemrosesan input dan output program, distance.py untuk kalkulasi dan pemilihan pasangan titik dengan jarak terdekat, dan visualizer.py untuk membuat visualisasi titik-titik pada bidang 2D dan 3D. Untuk menjalankan program utama, ketik perintah ".\run.bat" pada terminal folder "bin" dari repositori program.

Adapun alur program dimulai dari meminta input dari pengguna berupa jumlah titik dan jumlah dimensi. Setelah mendapatkan informasi tersebut, program akan menghasilkan titik-titik acak sebanyak jumlah titik masukan pengguna pada bidang dimensi sesuai masukan pengguna. Titik-titik acak tersebut merupakan titik bertipe bilangan real dan diacak dengan jangkauan $\left[-100(\frac{n}{10}+1), 100(\frac{n}{10}+1)\right]$. Jangkauan ini ditentukan sehingga semakin banyak titik yang dibangkitkan, jangkauan akan semakin besar sehingga distribusi titik-titik yang dibangkitkan menjadi lebih luas.

Setelah kumpulan titik-titik dibangkitkan, pasangan titik terdekat akan dicari dengan algoritma Brute Force dan juga algoritma Divide and Conquer yang telah dijabarkan pada bab sebelumnya. Kemudian, informasi dari hasil pencarian beserta dengan banyak operasi dan waktu eksekusinya ditampilkan ke layar.

Berikut adalah implementasi dari program pencarian pasangan titik terdekat dengan algoritma Divide and Conquer.

4.1.1 main.py

```
import time
import IO_processing
import visualizer
import distance

def main() :
```

```
    # Program Utama
    # Proses input
    IOresult = IO_processing.inputPoint()
    Point = IOresult[0]
    Pointdiv = Point.copy()
    n = IOresult[1]
    d = IOresult[2]

    # solve by brute force
    start = time.time()
    Hasil = distance.bruteforce(Point,n,d)
    end = time.time()
    print("Dengan strategi algoritma Brute Force,\npasangan titik terdekat adalah
titik ", end='')
    IO_processing.printPoint(Hasil[1],d)
    print(" dan titik ", end='')
    IO_processing.printPoint(Hasil[2],d)
    print(" dengan jarak %.4f" % (Hasil[0]))
    print("Waktu eksekusi:", (end-start)*1000, " ms")
    print("Banyak operasi: " + str(Hasil[3]))

    print("")

    # solve by divide and conquer
    start = time.time()
    Point2 = distance.quicksort(Pointdiv,0,len(Point)-1,0)
    Hasil = distance.divide_conquer(Point2,d,0)
    end = time.time()
    print("Dengan strategi algoritma Divide and Conquer,\npasangan titik terdekat
adalah titik ", end='')
    IO_processing.printPoint(Hasil[1],d)
    print(" dan titik ", end='')
    IO_processing.printPoint(Hasil[2],d)
    print(" dengan jarak %.4f" % (Hasil[0]))
    print("Waktu eksekusi:", (end-start)*1000, " ms")
    print("Banyak operasi: " + str(Hasil[3]))

    # visualizer
    visualizer.visualization(Point,n,d,Hasil[1],Hasil[2],Hasil[0])
```

```
main()
```

4.1.2 IO_processing.py

```
import random

def pembentukan_titik(n,d):
    # randomizer titik
    Point = [[round(random.uniform(-100*(n//10+1),100*(n//10+1)), 2) for c in range
(d)]for r in range (n)]
    #Point = [[random.randint(-100*(n//10),100*(n//10))for c in range (d)]for r in
range (n)]
    return Point

def printPoint(P,d):
    # print point (x, y, z, ...)
    print("(", end='')
    for j in range (0,d):
        print(P[j], end = '')
        if (j != d-1):
            print(", ",end='')
    print(")", end='')

def inputPoint():
    # input banyak titik, dimensi, dan generate points
    n = int(input("Masukkan banyak titik: "))
    d = int(input("Masukkan dimensi: "))
    print('')

    # validasi input
    if (n<=1):
        print("Jarak terdekat tidak bisa dikalkulasi! Silakan masukkan kembali input
paling sedikit 2 titik!\n")
        return inputPoint()
    elif (d<=0):
        print("Dimensi tidak valid! Silakan masukkan kembali input!\n")
        return inputPoint()
    else:
```

```
        Point = pembentukan_titik(n,d)
        # print points
        print("Generating points...")
        for i in range (0,n):
            print("Point", (i+1), end=' ')
            printPoint(Point[i],d)
            print('')
        print('')

        hasil = []
        hasil.append(Point)
        hasil.append(n)
        hasil.append(d)
        return hasil
```

4.1.3 distance.py

```
import math

def distance(Point,p1,p2,d):
    # Point = list of Points, p1 = titik 1, p2 = titik2, d = dimensi
    # Mengembalikan jarak, koordinat titik1, koordinat titik2
    sum = 0
    idx1 = []
    idx2 = []
    hasil = []
    for i in range (0,d) :
        sum += (Point[p1][i]-Point[p2][i])**2
        idx1.append(Point[p1][i])
        idx2.append(Point[p2][i])
    hasil.append(math.sqrt(sum))
    hasil.append(idx1)
    hasil.append(idx2)
    return hasil

def bruteforce(Point,n,d):
    # Point = list of points, n = total titik, d = dimensi
    # Mengembalikan jarak terdekat, pasangan titik1 dan titik2 terdekat, banyak
operasi distance
```

```
    hasil = []
    count = 0
    minlist = distance(Point,0,1,d)
    min = minlist[0]
    count +=1
    idx1 = minlist[1]
    idx2 = minlist[2]
    if (n>=3) :
        for i in range (0,n) :
            for j in range (i+1,n):
                if(i!= 0 or j!=1):
                    temp = distance(Point,i,j,d)
                    count +=1
                    if temp[0]<min :
                        min = temp[0]
                        idx1 = temp[1]
                        idx2 = temp[2]
    hasil.append(min)
    hasil.append(idx1)
    hasil.append(idx2)
    hasil.append(count)
    return hasil

def quicksort(Point,low,high,d_now) :
    # Point = list of points, low = index awal, high = index akhir, d_now = dimensi
sekarang
    # Mengembalikan list of points yang terurut membesar berdasarkan d_now
    # pi = pivot
    if low<high :
        pivot = Point[high][d_now]
        i = low -1
        for j in range (low,high):
            if Point[j][d_now]<=pivot:
                i +=1
                (Point[i],Point[j]) = (Point[j],Point[i])
        (Point[i+1],Point[high]) = (Point[high],Point[i+1])
        pi = i+1

        quicksort(Point,low,pi-1,d_now)
```

```
        quicksort(Point,pi+1,high,d_now)
    return Point

def checker (Point,min,d,p1,p2) :
    #Point = list of Points, min = jarak minimal, d = dimensi, p1 = titik1, p2 =
titik2
    #Mengembalikan boolean apakah harus dicek jaraknya/tidak
    summ = 0
    flag = True
    for i in range (0,d) :
        if (summ<min**2):
            summ += (Point[p1][i]-Point[p2][i])**2
        else :
            flag = False
            break
    return flag

def closest(Point,min,d,idx1,idx2):
    # Point = list of Points, min = jarak minimal, d = dimensi, idx1 = titik1 min,
idx2 = titik2 min
    # Mengembalikan jarak terdekat, pasangan titik1 dan titik2 jarak terdekat,
jumlah pemanggilan distance
    hasil = []
    count = 0
    for j in range (0,len(Point)):
        for k in range (j+1,len(Point)):
            flag = True
            for i in range (0,d):
                if abs(Point[j][i] - Point[k][i])>min :
                    flag = False
            if (flag) :
                Flag2 = True
                if (d>3) :
                    flag2 = checker(Point, min, d, j, k)
                if (flag2):
                    count +=1
                    temp = distance(Point,j,k,d)
                    if (temp[0]<min) :
                        min = temp[0]
```

```
                        idx1 = temp[1]
                        idx2 = temp[2]

    hasil.append(min)
    hasil.append(idx1)
    hasil.append(idx2)
    hasil.append(count)
    return hasil

def divide_conquer(Point,d,count):
    # Point = list of points, d = dimensi, count = banyak dipanggilnya distance
    # Mengembalikan jarak terdekat, pasangan titik1 dan titik2 jarak terdekat,
jumlah pemanggilan distance
    # prekondisi: Point terurut membesar
    hasil = []
    countbrute = 0
    if (len(Point)>=2 and len(Point)<=3): # basis
        if (len(Point)==2):
            countbrute=1
        else :
            countbrute=3
        return bruteforce(Point,len(Point),d)

    mid = len(Point)//2
    # menghitung jarak titik-titik dekat perbatasan
    midpoint = (Point[mid-1][0]+Point[mid][0])/2 # titik tengah antara left & right
    PointLeft = Point[:mid]
    PointRight = Point[mid:]

    # cari jarak minimum di area left dan right (rekurens)
    left = divide_conquer(PointLeft,d,count)
    right = divide_conquer(PointRight,d,count)
    if (left[0]<=right[0]):
        min = left[0]
        idx1 = left[1]
        idx2= left[2]
    else:
        min = right[0]
        idx1 = right[1]
```

```
        idx2= right[2]

    countleft = left[3]
    countright = right[3]

    # cari jarak terdekat di sekitar midpoint
    Pointcenter = []
    for i in range (0,len(PointLeft)):
        if abs(PointLeft[i][0] - midpoint)<=min :
            Pointcenter.append(PointLeft[i])
    for i in range (0,len(PointRight)):
        if abs(PointRight[i][0] - midpoint)<=min :
            Pointcenter.append(PointRight[i])
    minbaru = closest(Pointcenter,min,d,idx1,idx2)
    if (minbaru[0]<min):
        min = minbaru[0]
        idx1= minbaru[1]
        idx2= minbaru[2]

    countminbaru= minbaru[3]
    count = count + countleft + countright + countbrute + countminbaru

    hasil.append(min)
    hasil.append(idx1)
    hasil.append(idx2)
    hasil.append(count)
    return hasil
```

4.1.4 visualizer.py

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

def visualization(Point, n, d, Hasil1, Hasil2, min_distance):
    # Visualisasi untuk 2D dan 3D
    fig = plt.figure()

    if (d==2):
```

```
        # Visualisasi 2D
        ax = plt.axes()
        ax.set_xlabel('X-axis')
        ax.set_ylabel('Y-axis')

        for i in range (0,n):
            ax.scatter(Point[i][0], Point[i][1])
            ax.text(Point[i][0], Point[i][1], '%s' % (str(i)), size=8)

        # plot garis antara sepasang titik terdekat dan cetak jaraknya
        Hasil_x = []
        Hasil_x.append(Hasil1[0])
        Hasil_x.append(Hasil2[0])
        Hasil_y = []
        Hasil_y.append(Hasil1[1])
        Hasil_y.append(Hasil2[1])
        ax.plot(Hasil_x,Hasil_y)
        ax.text((Hasil1[0]+Hasil2[0])/2,(Hasil1[1]+Hasil2[1])/2, 'distance = %s' %
(str(round(min_distance,4))), size=6)

        plt.show()

    elif (d==3):
        # Visualisasi 3D
        ax = plt.axes(projection='3d')
        ax.set_xlabel('X-axis')
        ax.set_ylabel('Y-axis')
        ax.set_zlabel('Z-axis')

        for i in range (0,n):
            ax.scatter(Point[i][0], Point[i][1], Point[i][2])
            ax.text(Point[i][0], Point[i][1], Point[i][2], '%s' % (str(i)), size=8)

        # plot garis antara sepasang titik terdekat dan cetak jaraknya
        Hasil_x = []
        Hasil_x.append(Hasil1[0])
        Hasil_x.append(Hasil2[0])
        Hasil_y = []
        Hasil_y.append(Hasil1[1])
```

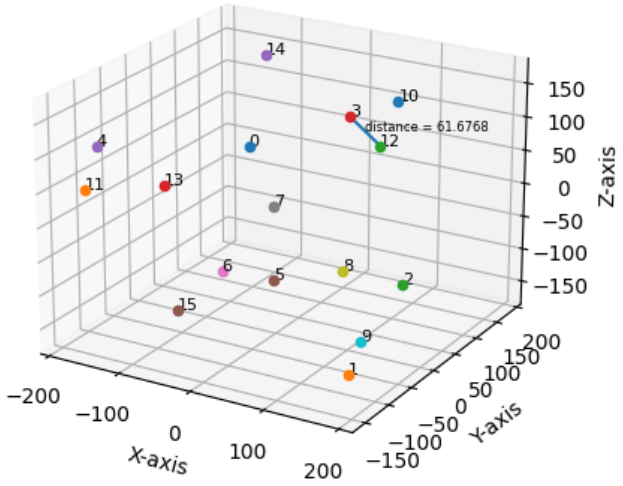
```
        Hasil_y.append(Hasil2[1])
        Hasil_z = []
        Hasil_z.append(Hasil1[2])
        Hasil_z.append(Hasil2[2])
        ax.plot(Hasil_x,Hasil_y,Hasil_z)

ax.text((Hasil1[0]+Hasil2[0])/2,(Hasil1[1]+Hasil2[1])/2,(Hasil1[2]+Hasil2[2])/2,
'distance = %s' % (str(round(min_distance,4))), size=6)

        plt.show()
```

4.2 Pengujian Program

4.2.1 Pengujian N = 16 pada bidang 3D

```
PS D:\kuliah\smt4\IF2211_Strategi_Algoritma\Tucil2_13521059_13521143\src> python main.py
Masukkan banyak titik: 16
Masukkan dimensi: 3

Generating points...
Point 1 (57.35, -139.8, 162.93)
Point 2 (190.61, -142.54, -127.44)
Point 3 (189.49, -25.17, -54.26)
Point 4 (0.86, 160.9, 75.13)
Point 5 (-169.65, -116.98, 105.3)
Point 6 (97.64, -152.0, -9.85)
Point 7 (-130.12, 77.39, -158.69)
Point 8 (-5.59, 4.47, 2.43)
Point 9 (103.42, -17.26, -57.62)
Point 10 (124.98, -9.4, -162.49)
Point 11 (101.23, 107.7, 138.75)
Point 12 (-184.93, -121.93, 39.27)
Point 13 (29.74, 185.32, 26.41)
Point 14 (-98.69, -83.97, 50.35)
Point 15 (-122.84, 164.64, 141.58)
Point 16 (-70.16, -101.25, -119.65)

Dengan strategi algoritma Brute Force,
pasangan titik terdekat adalah titik (0.86, 160.9, 75.13) dan titik (29.74, 185.32, 26.41) dengan jarak 61.6768
Waktu eksekusi: 0.0  ms
Banyak operasi: 120

Dengan strategi algoritma Divide and Conquer,
pasangan titik terdekat adalah titik (0.86, 160.9, 75.13) dan titik (29.74, 185.32, 26.41) dengan jarak 61.6768
Waktu eksekusi: 0.0  ms
Banyak operasi: 16
```

Gambar 4.2.1.1 Hasil Eksekusi Program untuk Pengujian 16 Titik pada Bidang 3 Dimensi



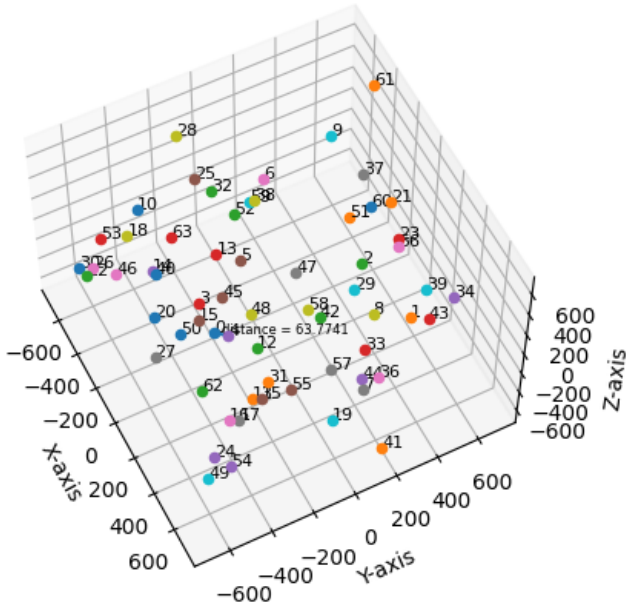
Gambar 4.2.1.2 Visualisasi untuk Pengujian 16 Titik pada Bidang 3 Dimensi

4.2.2 Pengujian N = 64 pada bidang 3D

```
PS D:\kuliah\smt4\IF2211_Strategi_Algoritma\Tucil2_13521059_13521143\src> python main.py
Masukkan banyak titik: 64
Masukkan dimensi: 3

Generating points...
Point 1 (76.53, -315.48, -54.61)
Point 2 (436.55, 429.27, 57.03)
Point 3 (-201.76, 538.2, -614.22)
Point 4 (46.4, -368.94, 235.49)
Point 5 (124.61, -274.56, -45.61)
Point 6 (-281.98, -18.36, -222.5)
Point 7 (-542.39, 223.03, -50.56)
Point 8 (638.75, 100.62, -10.19)
Point 9 (187.81, 391.37, -320.77)
Point 10 (-329.31, 436.32, 526.48)
Point 11 (-375.38, -451.22, 500.5)
Point 12 (403.81, -296.43, -162.8)
Point 13 (189.2, -172.27, -151.57)
Point 14 (-257.75, -144.74, -9.29)
Point 15 (-438.94, -370.15, -291.61)
Point 16 (309.81, -492.16, 648.05)
Point 17 (622.0, -508.99, 245.31)
Point 18 (302.91, -313.17, -573.55)
Point 19 (-225.47, -561.19, 596.32)
Point 20 (554.25, 4.3, -398.48)
Point 21 (64.84, -584.16, 330.06)
Point 22 (-31.46, 577.88, 266.84)
Point 23 (-501.24, -653.48, -185.02)
Point 24 (-29.48, 625.57, -134.12)
Point 25 (566.59, -554.74, -206.36)
Point 26 (-551.76, -105.18, 214.76)
Point 27 (-613.74, -577.89, -375.2)
Point 28 (63.28, -588.44, -83.69)
Point 29 (-608.06, -158.22, 583.11)
Point 30 (102.52, 338.45, -175.84)
Point 31 (-602.32, -645.25, -284.93)
Point 32 (178.13, -113.65, -586.47)
Point 33 (-683.11, 30.14, -241.78)
Point 34 (684.39, 73.91, 507.0)
Point 35 (384.53, 673.3, -52.82)
```

```
Point 36 (494.67, -300.91, 10.34)
Point 37 (359.76, 333.76, -636.13)
Point 38 (-323.86, 601.72, 26.18)
Point 39 (-266.19, 45.18, 344.25)
Point 40 (161.68, 669.56, -371.69)
Point 41 (-285.09, -420.52, 0.19)
Point 42 (699.98, 173.64, -601.06)
Point 43 (324.79, 57.59, 197.18)
Point 44 (347.54, 578.52, -261.53)
Point 45 (545.35, 149.17, -123.57)
Point 46 (-333.72, -89.42, -624.61)
Point 47 (-104.4, -669.39, 531.49)
Point 48 (-281.52, 253.51, -587.11)
Point 49 (208.46, -209.08, 263.52)
Point 50 (617.65, -610.16, -292.83)
Point 51 (85.45, -477.82, 97.36)
Point 52 (-451.66, 610.51, -626.57)
Point 53 (-175.52, -95.85, 489.8)
Point 54 (-299.67, -649.35, 520.98)
Point 55 (601.5, -496.76, -303.43)
Point 56 (385.18, -104.96, -291.04)
Point 57 (111.58, 541.3, 103.77)
Point 58 (640.28, -52.64, 346.21)
Point 59 (-0.83, 171.55, -415.5)
Point 60 (-540.31, 152.38, -211.51)
Point 61 (124.68, 390.02, 642.28)
Point 62 (-412.47, 683.88, 675.12)
Point 63 (86.82, -389.26, -596.33)
Point 64 (-305.09, -329.64, 249.38)

Dengan strategi algoritma Brute Force,
pasangan titik terdekat adalah titik (76.53, -315.48, -54.61) dan titik (124.61, -274.56, -45.61) dengan jarak 63.7741
Waktu eksekusi: 6.013154983520508  ms
Banyak operasi: 2016

Dengan strategi algoritma Divide and Conquer,
pasangan titik terdekat adalah titik (76.53, -315.48, -54.61) dan titik (124.61, -274.56, -45.61) dengan jarak 63.7741
Waktu eksekusi: 1.0194778442382812  ms
Banyak operasi: 85
```

Gambar 4.2.2.1 Hasil Eksekusi Program untuk Pengujian 64 Titik pada Bidang 3 Dimensi



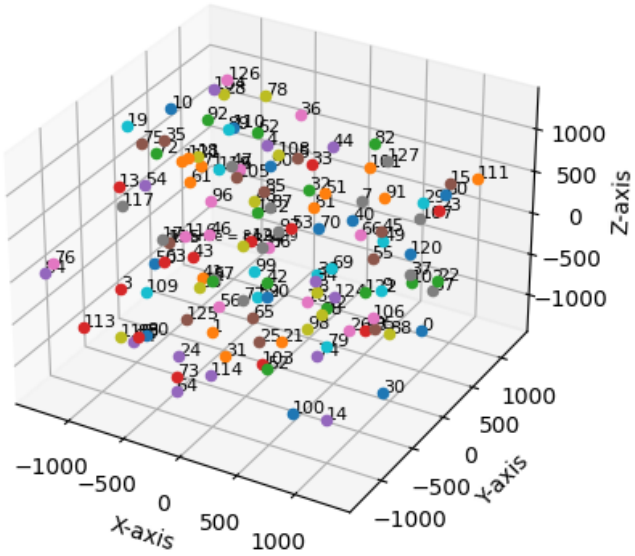
Gambar 4.2.2.2 Visualisasi untuk Pengujian 64 Titik pada Bidang 3 Dimensi

4.2.3 Pengujian N = 128 pada bidang 3D

```
Masukkan banyak titik: 128
Masukkan dimensi: 3

Generating points...
Point 1 (1200.33, 110.55, -588.24)
Point 2 (-488.85, -183.32, -1088.62)
Point 3 (-569.4, -45.2, -606.93)
Point 4 (-720.14, -1118.65, -11.51)
Point 5 (-800.92, 1195.12, 171.47)
Point 6 (88.73, 147.86, 979.13)
Point 7 (-947.75, 1013.36, -54.79)
Point 8 (695.83, 45.83, 782.3)
Point 9 (521.59, -314.48, -111.9)
Point 10 (384.41, 997.46, -1057.69)
Point 11 (-1197.92, 394.46, 959.66)
Point 12 (-1092.98, 378.83, 381.16)
Point 13 (155.36, 496.56, -1018.01)
Point 14 (-621.63, -1263.39, 1295.28)
Point 15 (1151.4, -1280.39, -663.16)
Point 16 (911.32, 1103.13, 377.35)
Point 17 (364.96, -189.64, -387.43)
Point 18 (-1292.91, 374.2, -657.04)
Point 19 (-734.25, 21.07, 788.67)
Point 20 (-1149.31, -321.57, 1211.92)
Point 21 (-341.68, 412.42, -1018.01)
Point 22 (528.35, -855.92, -311.59)
Point 23 (1254.73, 247.05, -70.9)
Point 24 (1029.72, 687.37, 370.63)
Point 25 (-269.86, -1048.0, -677.92)
Point 26 (163.96, -575.01, -668.85)
Point 27 (452.77, 325.04, -1043.75)
Point 28 (345.32, -830.73, 679.03)
Point 29 (-1108.36, 1048.74, 776.7)
Point 30 (1182.19, 108.89, 891.06)
Point 31 (1206.85, -514.67, -883.82)
Point 32 (-73.83, -684.97, -868.1)
Point 33 (570.05, -537.17, 1225.32)
```

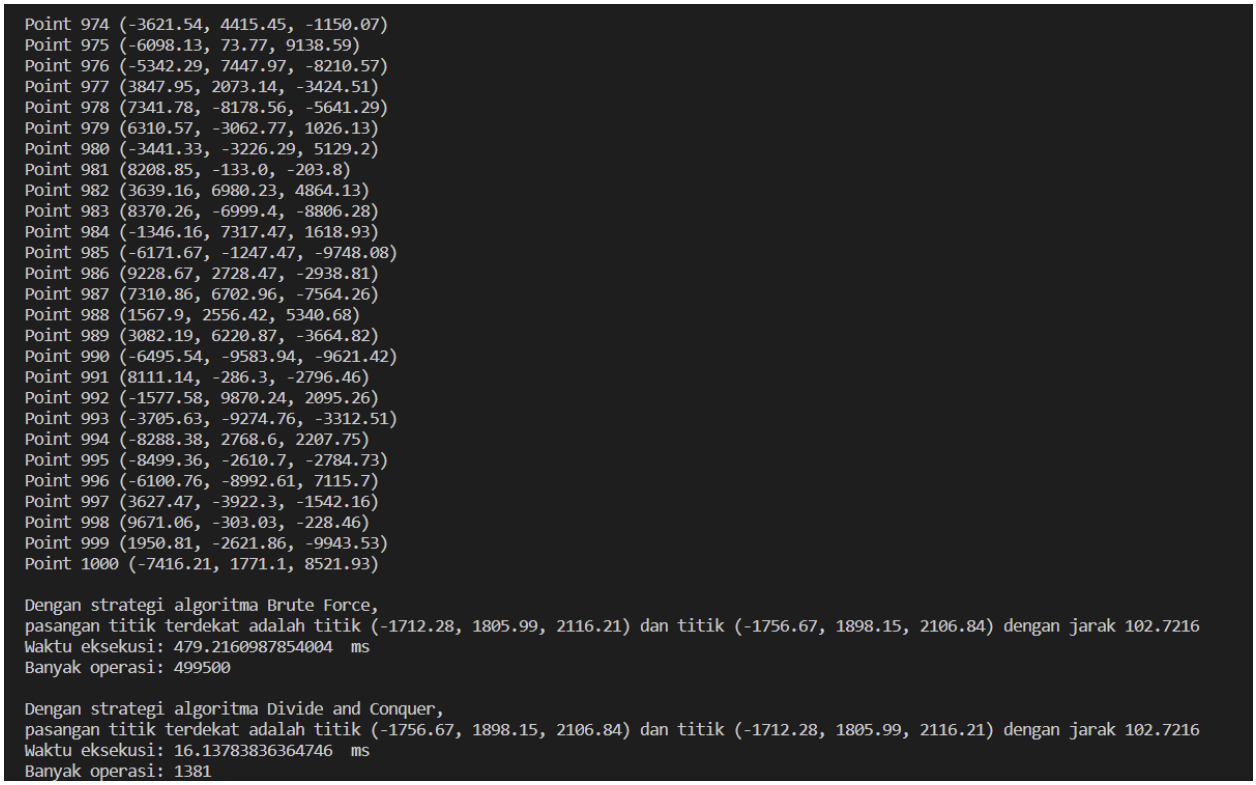
```
Point 102 (724.89, 114.27, 1119.66)
Point 103 (989.95, 343.89, -275.92)
Point 104 (-7.2, -232.45, -1263.38)
Point 105 (-916.5, 577.66, 1180.66)
Point 106 (-803.76, 736.22, 64.58)
Point 107 (1030.2, -361.25, -177.28)
Point 108 (1261.55, -72.38, 864.91)
Point 109 (-53.39, 93.62, 999.5)
Point 110 (-393.09, -1279.33, 186.18)
Point 111 (-916.37, 862.41, 555.27)
Point 112 (1277.73, 827.7, 753.04)
Point 113 (-504.36, 533.71, -131.07)
Point 114 (-1245.08, -873.05, -856.03)
Point 115 (-267.63, -583.45, -1250.9)
Point 116 (-1267.93, 434.95, -710.9)
Point 117 (-157.22, -1102.66, 798.57)
Point 118 (-1265.83, -247.8, 184.76)
Point 119 (-972.19, -761.23, -942.11)
Point 120 (-555.04, 38.53, 694.39)
Point 121 (821.75, 627.65, -176.83)
Point 122 (-680.94, -196.99, 916.97)
Point 123 (1129.3, -670.94, 372.55)
Point 124 (-392.9, 263.63, -258.07)
Point 125 (179.44, 582.1, -936.01)
Point 126 (-327.22, -822.5, -422.75)
Point 127 (-1130.11, 1143.54, 886.94)
Point 128 (645.06, 551.71, 894.19)

Dengan strategi algoritma Brute Force,
pasangan titik terdekat adalah titik (-1292.91, 374.2, -657.04) dan titik (-1267.93, 434.95, -710.9) dengan jarak 84.9439
Waktu eksekusi: 7.993936538696289  ms
Banyak operasi: 8128

Dengan strategi algoritma Divide and Conquer,
pasangan titik terdekat adalah titik (-1292.91, 374.2, -657.04) dan titik (-1267.93, 434.95, -710.9) dengan jarak 84.9439
Waktu eksekusi: 0.9989738464355469  ms
Banyak operasi: 160
```

Gambar 4.2.3.1 Hasil Eksekusi Program untuk Pengujian 128 Titik pada Bidang 3 Dimensi

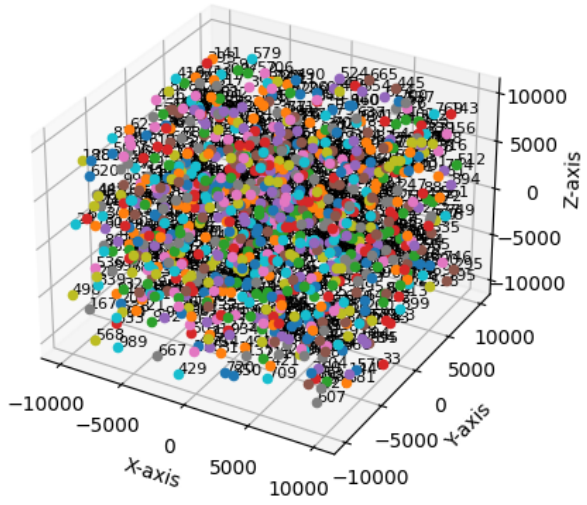


Gambar 4.2.3.2 Visualisasi untuk Pengujian 128 Titik pada Bidang 3 Dimensi

4.2.4 Pengujian N = 1000 pada bidang 3D



Gambar 4.2.4.1 Hasil Eksekusi Program untuk Pengujian 1000 Titik pada Bidang 3 Dimensi

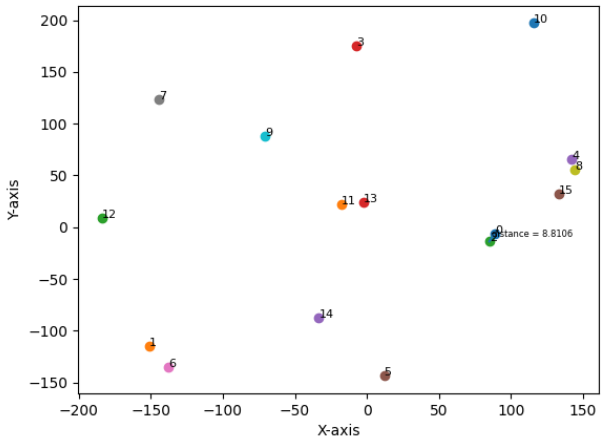


Gambar 4.2.4.2 Visualisasi untuk Pengujian 1000 Titik pada Bidang 3 Dimensi

4.2.5 Pengujian N = 16 pada bidang 2D

```
PS D:\kuliah\smt4\IF2211_Strategi_Algoritma\Tucil2_13521059_13521143\src> python main.py
Masukkan banyak titik: 16
Masukkan dimensi: 2

Generating points...
Point 1 (88.84, -5.91)
Point 2 (-150.88, -114.91)
Point 3 (84.98, -13.83)
Point 4 (-7.4, 174.81)
Point 5 (142.09, 66.18)
Point 6 (11.98, -143.7)
Point 7 (-137.88, -135.13)
Point 8 (-144.49, 123.24)
Point 9 (144.53, 55.47)
Point 10 (-70.88, 87.94)
Point 11 (115.68, 197.2)
Point 12 (-17.82, 21.74)
Point 13 (-184.07, 9.11)
Point 14 (-2.14, 24.1)
Point 15 (-33.32, -87.67)
Point 16 (133.0, 32.22)

Dengan strategi algoritma Brute Force,
pasangan titik terdekat adalah titik (88.84, -5.91) dan titik (84.98, -13.83) dengan jarak 8.8106
Waktu eksekusi: 1.0001659393310547  ms
Banyak operasi: 120

Dengan strategi algoritma Divide and Conquer,
pasangan titik terdekat adalah titik (84.98, -13.83) dan titik (88.84, -5.91) dengan jarak 8.8106
Waktu eksekusi: 0.0  ms
Banyak operasi: 13
```

Gambar 4.2.5.1 Hasil Eksekusi Program untuk Pengujian 16 Titik pada Bidang 2 Dimensi



Gambar 4.2.5.2 Visualisasi untuk Pengujian 16 Titik pada Bidang 2 Dimensi

4.2.6 Pengujian N = 64 pada bidang 1D

```
Masukkan banyak titik: 64
Masukkan dimensi: 1

Generating points...
Point 1 (-73.36)
Point 2 (-399.62)
Point 3 (-583.2)
Point 4 (68.94)
Point 5 (371.2)
Point 6 (-433.25)
Point 7 (360.09)
Point 8 (-117.04)
Point 9 (233.44)
Point 10 (296.34)
Point 11 (-683.63)
Point 12 (421.04)
Point 13 (5.07)
Point 14 (-615.85)
Point 15 (-689.79)
Point 16 (244.62)
Point 17 (-589.79)
Point 18 (-315.91)
Point 19 (-606.19)
Point 20 (-385.91)
Point 21 (-433.56)
Point 22 (619.32)
Point 23 (344.15)
Point 24 (342.74)
Point 25 (-232.88)
Point 26 (-33.6)
Point 27 (-687.22)
Point 28 (430.66)
Point 29 (-522.77)
Point 30 (531.27)
Point 31 (-269.27)
Point 32 (-587.96)
Point 33 (-107.03)
```

```
Point 38 (-107.47)
Point 39 (-507.68)
Point 40 (208.19)
Point 41 (390.71)
Point 42 (133.11)
Point 43 (-661.03)
Point 44 (277.48)
Point 45 (-213.21)
Point 46 (164.41)
Point 47 (-121.49)
Point 48 (488.06)
Point 49 (-685.07)
Point 50 (446.63)
Point 51 (256.55)
Point 52 (27.43)
Point 53 (-341.82)
Point 54 (-426.6)
Point 55 (36.25)
Point 56 (265.31)
Point 57 (205.93)
Point 58 (-85.51)
Point 59 (-686.44)
Point 60 (-629.82)
Point 61 (-344.44)
Point 62 (-486.69)
Point 63 (673.35)
Point 64 (-15.7)

Dengan strategi algoritma Brute Force,
pasangan titik terdekat adalah titik (-433.25) dan titik (-433.56) dengan jarak 0.3100
Waktu eksekusi: 1.9955635070800781  ms
Banyak operasi: 2016

Dengan strategi algoritma Divide and Conquer,
pasangan titik terdekat adalah titik (-433.56) dan titik (-433.25) dengan jarak 0.3100
Waktu eksekusi: 0.0  ms
Banyak operasi: 40
```

Gambar 4.2.6.1 Hasil Eksekusi Program untuk Pengujian 64 Titik pada Bidang 1 Dimensi

4.2.7 Pengujian N = 128 pada bidang 10D

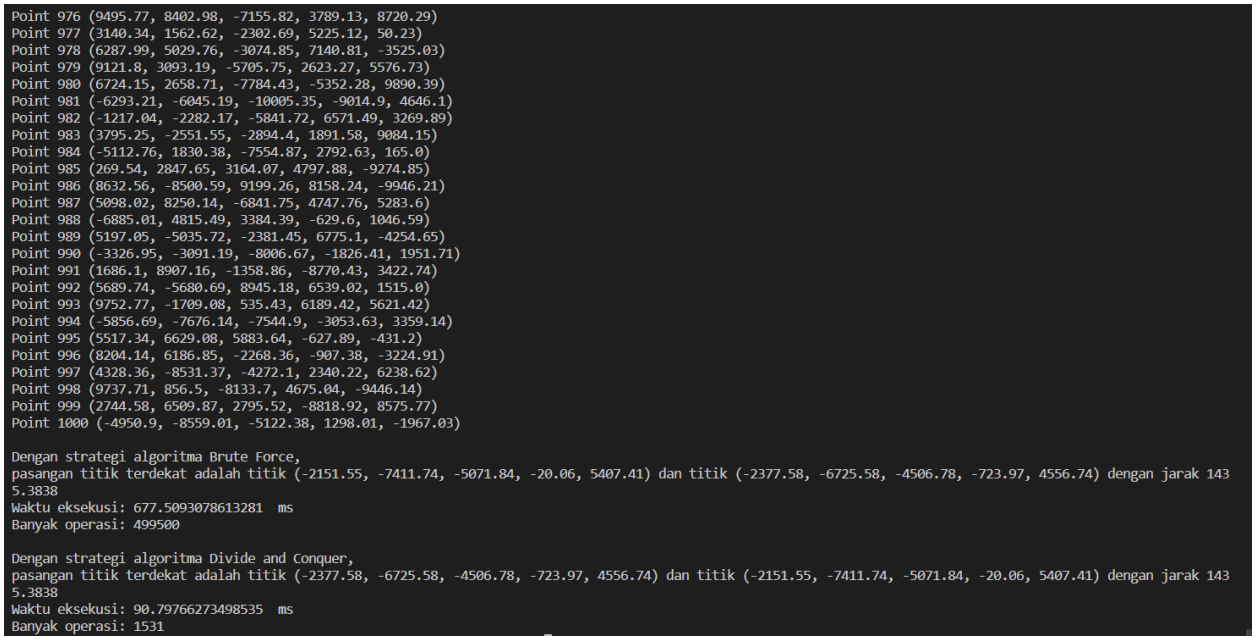
```
Masukkan banyak titik: 128
Masukkan dimensi: 10

Generating points...
Point 1 (-289.3, 711.3, -610.41, 78.56, -321.67, 1224.97, -1059.78, 817.64, -326.71, 1289.69)
Point 2 (1090.32, 653.7, 421.0, -875.59, 544.98, -488.27, 472.83, -758.24, 1081.77, 670.37)
Point 3 (-883.7, -35.62, 652.29, -563.97, 538.72, -1164.19, -1139.17, 933.44, -116.99, -1058.21)
Point 4 (-77.82, -1080.21, -382.43, -712.53, 364.9, 924.68, -324.84, 432.32, -765.93, 190.41)
Point 5 (-155.62, -892.99, -1005.78, 632.26, 63.48, 422.18, 169.63, -491.67, 603.41, -429.45)
Point 6 (-862.35, 414.9, 434.58, 1125.87, -958.95, 1258.57, -94.26, 415.43, 952.76, 133.28)
Point 7 (1277.02, 439.37, 147.17, 152.5, 367.17, 207.51, -728.82, 435.24, 1065.39, -30.48)
Point 8 (-870.72, 1209.11, -651.56, -507.99, 534.97, 871.4, -1162.23, 89.15, 1050.84, 794.22)
Point 9 (-473.05, 1289.77, 1036.0, -290.04, -505.23, 1170.67, 245.8, -93.14, 175.14, 125.49)
Point 10 (233.68, -741.07, 725.61, 972.38, 237.0, 374.94, -367.39, 345.34, 628.42, 235.87)
Point 11 (-832.85, 779.26, -1056.72, 939.8, -225.47, -936.39, 391.97, -13.72, -1185.57, 154.17)
Point 12 (-1288.79, -60.63, 611.56, -130.18, 1245.31, -487.35, -386.19, 928.62, 804.69, 931.62)
Point 13 (-478.01, -645.71, -44.82, -327.11, -481.51, -786.91, 973.66, 615.23, 765.09, 223.81)
Point 14 (379.91, 903.63, -338.36, 725.23, -289.12, -288.82, 951.0, 157.11, -745.72, -760.22)
Point 15 (202.95, 945.38, -1185.74, 1233.97, 807.07, -153.05, -821.32, 1117.68, 171.16, 537.73)
Point 16 (-300.22, 1054.19, -787.95, -279.24, 24.22, 114.79, -1036.7, 1296.84, 80.6, 969.1)
Point 17 (692.18, -486.77, 726.05, 89.71, 1271.97, 115.7, 764.13, -971.68, 140.33, 1164.54)
Point 18 (1256.3, -344.42, -1279.3, -889.97, 1165.32, -1214.27, -645.83, 403.83, -93.83, -1136.29)
Point 19 (16.22, 1259.09, 511.01, 449.1, -214.64, 264.26, 255.37, -38.98, -994.2, -708.98)
Point 20 (1097.84, 1232.45, -631.94, 253.07, 1158.13, -581.93, -100.82, -1024.11, -1244.1, 777.52)
Point 21 (1271.95, 195.11, 17.91, -463.49, -1053.34, -971.95, -592.12, -750.28, -723.63, 1088.55)
Point 22 (-1137.62, -185.01, 573.92, 100.11, 409.3, -772.16, 519.54, -1184.62, -909.85, 877.68)
Point 23 (-366.61, 1210.48, -519.4, -833.96, -235.35, -611.21, -1255.76, 220.55, -125.53, -285.36)
Point 24 (-1241.9, -323.08, 900.02, 152.79, -1085.0, -75.67, 758.42, -1081.34, 1071.97, 907.0)
Point 25 (-1202.25, -545.45, 345.28, -1071.35, -650.1, -742.87, 410.41, -212.52, 1101.51, -1021.67)
Point 26 (-1282.63, -4.45, -188.93, 941.21, -334.88, 557.29, 957.76, 116.6, -1272.81, -306.21)
Point 27 (1043.84, 3.94, -880.26, -1041.86, -1026.67, -27.21, -1078.35, -753.07, -412.34, 175.07)
Point 28 (-36.74, -259.42, 839.88, 50.47, -32.9, 798.38, -1262.18, 103.41, -1085.72, -910.49)
Point 29 (522.34, -1018.62, -581.36, -526.54, -290.5, -953.57, 430.71, 734.04, -268.64, -48.66)
Point 30 (-220.43, 1260.51, -1048.98, 1118.97, -396.76, 527.14, -896.82, -821.05, 59.55, -1135.97)
Point 31 (715.41, 1007.49, 439.69, 585.73, -530.14, -1148.69, 589.48, -454.56, -468.97, -1080.8)
Point 32 (558.13, -235.18, 531.42, 1101.62, 826.29, 6.08, 738.81, -582.49, -1098.05, -18.45)
```

```
Point 104 (-389.58, -1117.91, 652.09, -1053.92, -839.55, -805.41, -122.73, -1080.71, 76.81, -606.13)
Point 105 (-26.11, 548.29, -489.74, -979.86, -934.06, -617.79, 1260.62, -1110.09, -827.92, 559.1)
Point 106 (957.54, 273.62, 384.71, 967.67, 770.46, -742.51, -333.26, -629.54, -427.44, -841.95)
Point 107 (-1285.04, 1221.89, 35.55, -1064.99, 213.13, -764.19, -576.67, 935.25, 165.8, 264.98)
Point 108 (833.27, 1065.14, -1200.14, -104.41, -793.7, -597.71, -652.12, -347.95, 736.37, -808.97)
Point 109 (717.93, -983.29, 1053.28, 143.4, 402.45, -965.64, -410.45, 22.27, -39.78, 423.35)
Point 110 (120.99, -691.89, -1287.33, -735.29, 701.71, -14.99, 211.98, 1221.45, -6.51, 696.39)
Point 111 (604.26, -851.12, -618.57, -34.28, 1084.96, 628.57, 156.29, -662.03, -849.05, 649.94)
Point 112 (-1033.77, -815.0, 851.13, 512.66, -1051.28, 197.38, 226.11, -622.07, 575.87, 649.67)
Point 113 (-1063.79, -1163.17, -607.48, 164.14, -116.28, 662.86, 584.05, -482.87, -297.86, 1078.29)
Point 114 (553.23, -1123.82, -144.31, 1075.76, -713.4, 817.6, 769.0, -871.39, 719.88, 506.51)
Point 115 (-1034.85, 506.41, 269.96, 1263.36, 1207.36, 1211.11, -906.96, 1242.84, 816.17, 365.99)
Point 116 (1294.35, 138.34, 459.21, 280.11, 789.2, -653.39, -609.81, -318.07, -684.6, -757.7)
Point 117 (-771.43, 601.24, 295.35, -755.82, -606.62, 345.31, -451.5, -870.98, -711.87, 795.85)
Point 118 (-928.27, -1273.38, 589.93, -756.48, 539.14, 267.88, 100.74, -15.94, 399.07, 929.98)
Point 119 (-516.91, 966.87, -338.19, 1285.06, -399.8, 290.5, -72.52, 879.94, 659.51, -1051.41)
Point 120 (912.87, 679.86, -1141.51, -353.6, 48.07, 342.92, 1248.56, -384.51, -1193.96, -287.41)
Point 121 (259.35, -652.73, -356.76, -1000.33, -77.07, 1055.18, 1013.79, 977.76, 1195.57, 1222.86)
Point 122 (449.67, -1212.28, -1087.11, 961.78, 181.76, -31.66, -1083.29, 1099.31, -867.15, -941.27)
Point 123 (-782.04, -42.21, 98.64, 1218.47, -1200.87, 940.25, -932.58, -772.04, 898.46, 976.02)
Point 124 (-849.88, -125.1, 584.87, -1223.14, 605.36, 1015.98, 105.13, -1198.37, -805.65, 630.14)
Point 125 (132.23, 1184.02, 165.15, 172.44, -1096.19, -1245.25, 617.46, 1236.75, -1121.04, -524.34)
Point 126 (-700.92, 664.82, 1116.23, -1270.4, 1024.26, 257.45, -1067.13, -484.53, 808.72, -233.35)
Point 127 (281.25, 703.32, 1090.98, 1297.84, -1124.16, 507.73, 420.08, -450.78, -949.68, -197.99)
Point 128 (-374.42, 397.17, -594.39, -337.97, -188.11, 363.24, -1059.81, 504.35, 846.85, 681.02)

Dengan strategi algoritma Brute Force,
pasangan titik terdekat adalah titik (957.54, 273.62, 384.71, 967.67, 770.46, -742.51, -333.26, -629.54, -427.44, -841.95) dan titik (1294.35, 138.34, 459.21, 2
80.11, 789.2, -653.39, -609.81, -318.07, -684.6, -757.7) dengan jarak 930.0790
Waktu eksekusi: 18.000364303588867  ms
Banyak operasi: 8128

Dengan strategi algoritma Divide and Conquer,
pasangan titik terdekat adalah titik (957.54, 273.62, 384.71, 967.67, 770.46, -742.51, -333.26, -629.54, -427.44, -841.95) dan titik (1294.35, 138.34, 459.21, 2
80.11, 789.2, -653.39, -609.81, -318.07, -684.6, -757.7) dengan jarak 930.0790
Waktu eksekusi: 10.533571243286133  ms
Banyak operasi: 194
```

Gambar 4.2.7.1 Hasil Eksekusi Program untuk Pengujian 128 Titik pada Bidang 10 Dimensi

4.2.8 Pengujian N = 1000 pada bidang 5D

```
Masukkan banyak titik: 1000
Masukkan dimensi: 5

Generating points...
Point 1 (-110.4, -261.31, 2573.02, -5536.94, 5355.25)
Point 2 (-7484.86, 808.97, 2834.57, 9579.25, -415.0)
Point 3 (-9601.96, 9045.93, -3444.72, 5743.81, -6828.54)
Point 4 (-485.14, -834.55, -8340.63, -3363.29, 4358.4)
Point 5 (-3116.98, -162.82, -985.22, -2545.8, -7527.53)
Point 6 (-9685.14, -964.71, 938.62, 2402.65, -5961.03)
Point 7 (-9795.27, -1792.66, 1327.89, 9186.04, -4578.11)
Point 8 (637.33, 4757.79, 7938.31, 7725.71, -8593.62)
Point 9 (-4750.25, 7058.98, -6005.12, 7589.95, 7007.64)
Point 10 (6721.01, -1488.91, -5871.3, -4927.35, -10056.23)
Point 11 (-3485.8, 52.15, 8594.11, 1756.69, 880.83)
Point 12 (-3005.96, 7727.13, 324.38, 6184.03, -8855.26)
Point 13 (166.24, -5718.63, 9362.31, -1630.7, 2103.04)
Point 14 (-3990.73, 5160.97, 8080.64, -9761.53, -2664.49)
Point 15 (-3461.32, 5927.66, -9954.79, -1289.62, -8166.94)
Point 16 (-6218.49, -621.4, 3658.52, 9549.76, -215.64)
Point 17 (3912.07, -8330.89, 7205.05, 1990.05, -8160.16)
Point 18 (10037.71, -6647.05, 8015.41, 1820.12, 10024.63)
Point 19 (6771.77, 8572.47, -2427.82, 4638.99, 1833.52)
Point 20 (1769.54, 6658.45, -297.49, -7266.87, -1222.4)
Point 21 (-7985.82, 373.98, -1846.81, -4878.75, -8180.48)
Point 22 (-5715.67, 1249.44, -2171.55, -6174.13, 7804.73)
Point 23 (7616.5, 7405.31, -5632.62, -3799.19, 2212.72)
Point 24 (2897.88, -1461.92, 2126.61, -5996.27, -6348.47)
Point 25 (4912.83, -9635.37, 1451.99, -5682.23, 7185.78)
Point 26 (1546.31, -3783.19, -3726.13, -934.56, 1625.84)
Point 27 (4922.6, -458.66, -2921.64, -7108.24, -473.4)
Point 28 (-2480.26, -3676.98, 586.03, 8908.47, -7708.49)
Point 29 (7946.41, -8652.97, 9691.13, -229.1, -6010.95)
Point 30 (-4574.89, 5402.8, 7512.07, -6526.26, -7503.11)
Point 31 (4257.22, -9402.94, 4217.73, 2267.05, 9225.16)
Point 32 (9266.29, 6300.46, 8702.56, -4378.53, -4823.1)
```



Gambar 4.2.8.1 Hasil Eksekusi Program untuk Pengujian 1000 Titik pada Bidang 5 Dimensi

BAB V
PENUTUP

5.1 Simpulan

Melalui Tugas Kecil 2 IF2211 Strategi Algoritma, kami berhasil membuat sebuah program untuk mencari pasangan titik terdekat pada bidang tiga dimensi maupun n dimensi dengan menggunakan algoritma Divide and Conquer dan algoritma Brute Force. Untuk keberjalanan program, kami juga memanfaatkan Quicksort sebagai algoritma pengurutan (*sorting*) dengan menggunakan algoritma Divide and Conquer, serta menggunakan rumus Euclidean Distance untuk menghitung jarak antara dua buah titik.

Dengan membandingkan kedua algoritma yang telah dibuat, dapat dilihat bahwa algoritma Divide and Conquer lebih efektif dan efisien jika dibandingkan dengan algoritma Brute Force untuk dimensi rendah. Hal ini dapat dilihat dari kompleksitas waktu pada algoritma Divide and Conquer yang menghasilkan $O(n\ (\ log\ n)^{d-1})$ dan algoritma Brute Force sebesar $O(n^2)$.

DAFTAR REFERENSI

Matplotlib. “Matplotlib 3.7.0 Documentation” https://matplotlib.org/stable/index.html (Diakses 28 Februari 2023).

Munir, Rinaldi. Sekolah Teknik Elektro dan Informatika ITB. 2023. “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 28 Februari 2023).

Munir, Rinaldi. Sekolah Teknik Elektro dan Informatika ITB. 2023. “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 28 Februari 2023).

Python. “Python Documentation”. https://www.python.org/doc/ (Diakses 28 Februari 2023).

Suri, Subhash. University of California, Santa Barbara, Computer Science. “Closest Pair”. https://sites.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf (Diakses 28 Februari 2023).

LAMPIRAN

Lampiran 1: Tautan Repositori GitHub

Tautan repositori GitHub Tugas Kecil 2 IF2211 Strategi Algoritma:

https://github.com/arleenchr/Tucil2_13521059_13521143

Lampiran 2: Tabel Penilaian

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