

LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA
PENCARIAN PASANGAN TITIK TERDEKAT PADA
BIDANG 3D DENGAN ALGORITMA *DIVIDE AND*
CONQUER



Disusun Oleh

Salomo Reinhart Gregory Manalu (13521063)

Michael Leon Putra Widhi (13521108)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2023

BAB I

ALGORITMA *DIVIDE AND CONQUER*

A. Algoritma *Divide and Conquer* dalam pencarian solusi pasangan titik terdekat

Persoalan pencarian solusi pasangan titik terdekat atau *Closest-Pair Problems*, sesuai dengan namanya, memiliki tujuan untuk menentukan sepasang titik dari himpunan titik P yang terdiri atas n titik yang jaraknya terdekat satu dengan yang lain. Misalkan terdapat dua buah titik 2 dimensi, maka jarak dari kedua titik tersebut dapat ditentukan dengan menggunakan persamaan jarak Euclidean sebagai berikut.

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

Dengan d menyatakan nilai dari jarak euclidean dua buah titik yang memiliki koordinat kartesian (x_1, y_1) dan (x_2, y_2) . Persamaan ini dapat digeneralisasi untuk menghitung jarak dari dua titik pada dimensi yang lebih umum, seperti pada ruang bidang R^n .

Masalah ini muncul di sejumlah aplikasi. Misalnya, dalam kontrol lalu lintas udara, Persoalan ini dapat memantau lokasi pesawat yang terlalu berdekatan untuk mengindikasikan kemungkinan tabrakan.

Pada Tugas Kecil ini, kita diminta untuk mengembangkan algoritma pencarian sepasang titik terdekat pada bidang 3 dimensi dengan menggunakan jarak euclidean yang di ekstensi bagi titik pada bidang 3 dimensi pula. Persoalan ini akan diselesaikan dengan Algoritma *Divide and Conquer* dan hasilnya akan dibandingkan dengan Algoritma *Brute Force*. Dengan menggunakan Algoritma *Divide and Conquer* kita dapat menyelesaikan masalah pencarian pasangan titik terdekat dengan lebih mangkus dan sangkil, tetapi tetap memperoleh hasil yang tepat.

B. Penjelasan Algoritma Program

1. Implementasi solusi dalam Algoritma *Divide and Conquer*

Secara umum alur jalannya Algoritma *Divide and Conquer* untuk menyelesaikan masalah pencarian titik terdekat adalah sebagai berikut :

- a. Misalkan P adalah senarai yang terdiri atas n buah titik. Jika nilai $2 \leq n \leq 3$, maka selesaikan secara langsung dengan memanggil persamaan jarak euclidean. Jika $n > 3$, maka urutkan senarai tersebut berdasarkan nilai absis.
- b. Partisi kumpulan titik dalam senarai tersebut menjadi dua bagian, misal P_1 dan P_2 sama besar. Proses partisi dapat dilakukan dengan membuat garis vertikal semu yang membagi dua wilayah melalui nilai median m dari nilai absis yang sudah diurutkan, sehingga akan ada $\lfloor n/2 \rfloor$ titik yang terletak di bagian kiri dan $\lfloor n/2 \rfloor$ titik yang terletak di bagian kanan.
- c. Tentukan jarak terpendek dari kedua upa-senarai hasil partisi secara rekursif, misalkan d_1 dan d_2 , dan nyatakan nilai terkecil d sedemikian hingga $d = \min\{d_1, d_2\}$. Perlu dicatat bahwa d bukanlah jarak terpendek antara kedua titik karena bisa saja pasangan titik dengan jarak yang lebih dekat terletak pada sekitar garis vertikal semu. Melalui ketiga langkah di atas, telah ditentukan batas atas dari jarak terkecil yang mungkin dari kumpulan titik dalam senarai.
- d. Dengan mengetahui nilai batas atas d , dapat dilakukan pencarian pasangan titik terdekat pada daerah sekitar garis vertikal semu yang mempartisi dua wilayah dengan batas jelajah sebesar d pada bagian kanan dan kiri garis semu. Tidak perlu melakukan kalkulasi seluruh titik pada wilayah tersebut, cukup titik-titik yang jarak masing-masing komponen koordinat konstruktornya bernilai lebih kecil dari d .
- e. Langkah terakhir adalah melakukan kombinasi dari ketiga hasil yang telah diperoleh sebelumnya untuk kemudian ditentukan nilai yang paling kecil dari ketiganya, yaitu pasangan titik dengan jarak terpendek pada wilayah partisi kanan, partisi kiri, dan kumpulan titik di wilayah sekitar garis semu vertikal.

Mengembangkan permasalahan ini ke dalam dimensi 3, alih-alih membayangkan garis vertikal semu yang mempartisi dua wilayah, bayangkan terdapat sebuah plat super besar dengan lebar d berpusat pada $(0,0,0)$ dengan lebar dan kedalaman yang tak-berhingga, sejajar terhadap sumbu- y dan z pada ruang kartesian.

Setelah melakukan partisi dengan “plat” tersebut, akan terdapat dua buah region kubus super besar yang mengandung sekiranya $\lfloor n/2 \rfloor$ titik pada mega-balok kiri dan

$\lfloor n/2 \rfloor$ titik yang terletak pada mega-balok kanan. Pencarian pasangan titik dengan jarak terdekat dilakukan secara rekursif pada kedua wilayah partisi hingga ditemukan jarak terkecil dari masing-masing wilayah untuk kemudian dibandingkan dan diambil jarak terkecilnya.

Lebih lanjut, jarak terkecil tersebut digunakan sebagai batas atas daerah jelajah dari lebar plat partisi pada wilayah mega-balok kanan dan kiri. Perlu diingat kembali bidang partisi ini merupakan bidang 2 dimensi, sehingga dapat digunakan algoritma pencarian pasangan titik terdekat pada wilayah 2 dimensi untuk menentukan pasangan titik terdekat pada plat partisi ini.

Terakhir, sekaligus yang paling penting, adalah membandingkan nilai jarak dari tiga kemungkinan wilayah jelajah untuk diambil nilai jarak terkecilnya.

2. Implementasi solusi pembanding dengan Algoritma *Brute Force*

Secara umum alur jalannya Algoritma *Brute Force* untuk menyelesaikan masalah pencarian titik terdekat adalah sebagai berikut :

- a. Misalkan P adalah senarai yang terdiri atas n buah titik. Nilai jarak terpendek diinisialisasi dengan jarak antara titik pertama dan titik kedua. Nilai jarak terpendek ini dicari dengan menggunakan fungsi *EuclideanDist3* (untuk 3 dimensi) ataupun *EuclideanDistGeneral*.
- b. Setelah itu, dilakukan pencarian jarak antara setiap titik pada senarai dengan setiap titik lainnya. Nilai jarak yang didapat ini akan dibandingkan dengan nilai jarak terpendek yang telah diinisialisasi pada langkah pertama.
- c. Misalkan a adalah nilai jarak antara titik ke-2 dan ke-4. *Shortest* adalah variabel yang menyimpan nilai jarak terpendek sementara. Apabila a lebih kecil daripada *Shortest*, maka nilai *Shortest* diisi oleh nilai a ($Shortest = a$). Apabila a lebih besar daripada *Shortest*, langkah selanjutnya adalah membandingkan jarak titik ke-2 dan ke-5 dengan nilai *Shortest*. Algoritma ini dilakukan hingga setiap titik selesai dibandingkan jaraknya dengan titik lainnya.

C. Analisis Kompleksitas Algoritma Program

1. Kompleksitas hasil implementasi Algoritma *Divide and Conquer*

Solusi yang diimplementasikan dalam algoritma ini adalah secara linear, membagi permasalahan menjadi dua bagian dengan ukuran yang sama besar atau mendekati sama besar dan menggabungkan hasilnya untuk memperoleh sebuah solusi yang utuh. Dengan mengasumsikan jumlah elemen n merupakan sebuah bentuk hasil perpangkatan 2 dan proses pengurutan tidak diperhitungkan dalam penentuan kompleksitas algoritma, maka kompleksitas waktu dari algoritma ini dapat dinyatakan dengan persamaan berikut.

$$T(n) = 2T(n/2) + cn, \quad n \geq 3$$

dengan ekspresi $T(n/2)$ bentuk rekursifitas partisi permasalahan menjadi 2 upa-permasalahan yang seimbang dan cn menyatakan jumlah operasi yang dilakukan pada bagian plat partisi.

Dengan menggunakan teorema master ($a = 2$, $b = 2$, dan $d = 1$), maka dapat disimpulkan bahwa Algoritma *Divide and Conquer* yang diimplementasikan pada permasalahan ini memiliki kompleksitas $T(n) = O(n \log n)$.

2. Kompleksitas hasil pembandingan dengan Algoritma *Brute Force*

Penyelesaian masalah pencarian 2 titik terdekat dengan algoritma *Brute Force* adalah dengan cara membandingkan setiap titik dengan titik lainnya. Prosedur ini dilakukan dengan menggunakan 2 buah kalang bertingkat (*for loop*) saling berkait untuk melakukan iterasi yang lebih sangkil. Misalkan n adalah jumlah elemen. Maka, kompleksitas waktu dari algoritma ini adalah $T(n) = 1/2 n^2 = O(n^2)$.

D. Implementasi Bonus

1. Visualisasi bidang 3 Dimensi

Visualisasi bidang 3 dimensi dalam tugas kecil ini menggunakan *library* yang ada dalam bahasa pemrograman *Python*. *Library* tersebut adalah **matplotlib**. Berikut adalah langkah-langkah dalam membuat visualisasi bidang 3 dimensi :

- Langkah pertama adalah menciptakan sebuah *figure*
- Menggambar plot-plot dengan fungsi `add_subplot()`.
- Fungsi yang dibuat (*VisualizeMinimum*) akan menerima senarai yang berisi semua titik yang dibangkitkan dan 2 titik dengan jarak terdekat. Fungsi ini akan

melakukan *plotting* terhadap semua titik yang ada dalam senarai dan mewarnai semua titik dengan warna hitam. Dua titik lainnya yang memiliki jarak terpendek akan diwarnai merah.

2. Generalisasi masalah untuk titik pada bidang R^n

Poin penting dari penyelesaian permasalahan ini dalam algoritma *Divide and Conquer* adalah sama seperti yang telah dijabarkan sebelumnya. Membawa permasalahan ini ke sesuatu yang lebih umum seperti pada kasus bidang R^n memerlukan sebuah pendekatan ekstra untuk menyelesaikannya. Perkenalkan konsep *Sparsity Condition*, kondisi dimana sebuah balok dengan lebar 2δ pasti mengandung sebanyak $O(1)$ titik dari kumpulan titik P .

Mengingat kembali penyelesaian utama dari algoritma *Divide and Conquer* yang menyelesaikan permasalahan pada wilayah kanan dan kiri secara rekursif. Konsep sparsitas menyangkut penggabungan solusi kedua nilai terkecil tersebut dengan wilayah yang berada di sekitar pusat partisi dalam batas jelajah sebesar nilai terkecil dari kedua wilayah tersebut, sebut saja δ . Jika S adalah sebuah himpunan titik di sekitar wilayah pusat partisi dengan batas jelajah setidaknya δ , maka kubus- δ yang berpusat di titik pusat dimensi memiliki sebuah jumlah konstan dari titik-titik pada S , bergantung pada besarnya dimensi permasalahan.

Pembuktian dari kondisi tersebut dapat dijelaskan sebagai berikut. Misalkan terdapat permasalahan pencarian pasangan titik terdekat pada dimensi d dengan C adalah sebuah kubus- δ yang berpusat pada titik pusat dimensi permasalahan dan L adalah himpunan titik yang berada pada wilayah C . Bayangkan meletakkan bola dengan radius $\delta/2$ di sekitar setiap titik pada himpunan titik L . Maka dapat dinyatakan hasil sebagai berikut :

- a. Volume dari kubus C adalah $(2\delta)^d$
- b. Volume dari setiap bola yang diletakkan adalah $\frac{1}{C_d}(\delta/2)^d$ dengan nilai C_d yang konstan.

Melalui kedua fakta tersebut, maka jumlah maksimum dari bola (atau titik) pada sebuah wilayah adalah maksimum sebanyak $C_d 4^d$, atau memiliki jumlah konstan yaitu sebesar $O(1)$.

Dengan mengetahui konsep sparsitas, maka dapat dirumuskan algoritma pencarian titik terdekat pada bidang R^n sebagai berikut :

- Misalkan P adalah senarai yang terdiri atas n buah titik, partisi kumpulan titik dalam senarai tersebut menjadi dua bagian, misal P_1 dan P_2 sama besar dengan nilai median hiper-bidang yang normal dengan beberapa sumbu.
- Secara rekursif, cari penyelesaian pasangan titik dengan jarak terpendek, δ_1 dan δ_2 dari wilayah P_1 dan P_2 secara berurutan. Selanjutnya tentukan nilai δ sehingga $\delta = \min\{\delta_1, \delta_2\}$.
- Misalkan S' adalah sekumpulan titik yang berada di sekitar bidang pusat partisi H dengan daerah jelajah sebesar δ , langkah yang dapat dilakukan adalah memproyeksikan sekumpulan titik S' ke bidang pusat partisi H . Proyeksi ini ditujukan untuk memperkecil dimensi kasus pada daerah tersebut.
- Gunakan sifat kondisi sparsitas δ untuk memeriksa semua pasangan titik pada S' secara rekursif. Menurut hasil analisis yang sebelumnya telah dijabarkan, terdapat $O(n)$ pasangan titik saja di sekitar kubus- δ ini

Melalui susunan algoritma tersebut, maka rekurensi dari algoritma ini dapat dinyatakan kompleksitas waktunya sebagai berikut :

$$\begin{aligned} T(n, d) &= 2T(n/2, d) + U(n, d - 1) + O(n) \\ &= 2T(n/2, d) + O(n (\log n)^{d-2}) + O(n) \\ &= O(n (\log n)^{d-1}) \end{aligned}$$

Dengan T menyatakan kompleksitas waktu, U menyatakan kompleksitas penyelesaian kondisi sparsitas pada kubus- δ , dan d adalah dimensi permasalahan pada ruang bidang R^n . Perhatikan bahwa kompleksitas algoritma akan meningkat seiring dengan bertambahnya nilai dimensi permasalahan karena skema rekurensi yang bertingkat.

BAB II

SOURCE CODE PROGRAM

A. File Titik (Point.py)

Pada *file* ini didefinisikan semua fungsi dan prosedur yang berhubungan dengan pemrosesan terhadap titik dalam suatu bidang koordinat. Berikut adalah tampilannya.

1. Impor modul eksternal

```
# File pengimplementasian Pemrosesan Titik
# 0. Impor modul eksternal
import random
import math
```

Gambar 2.1.1. Modul eksternal yang diimpor untuk membantu jalannya program.

2. Fungsi RandomPoint

```
# 1. Pembangkit titik koordinat acak
def RandomPoint(dimension) :
    return [round(random.uniform(-1000, 1000), 2) for i in range (dimension)]
```

Gambar 2.1.2. Fungsi RandomPoint untuk membangkitkan sebuah titik koordinat acak berdasarkan dimensi dengan rentang nilai -1000 hingga 1000.

3. Fungsi ListRandomPoint

```
# 2. Inisialisasi senarai hasil pembangkit titik koordinat acak
def ListRandomPoint(dimension, numbers) :
    return [RandomPoint(dimension) for i in range (numbers)]
```

Gambar 2.1.3. Fungsi ListRandomPoint untuk membangkitkan sejumlah titik sebanyak *numbers* berdasarkan nilai dimensi masukan untuk disimpan dalam matriks berukuran 1 x dimensi.

4. Fungsi EuclideanDist3

```
# 3. Persamaan Euclidean distance (3 dimensi)
def EuclideanDist3(point1, point2) :
    # Persamaan : sqrt((x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2)
    xdist = pow(point1[0] - point2[0], 2)
    ydist = pow(point1[1] - point2[1], 2)
    zdist = pow(point1[2] - point2[2], 2)
    return math.sqrt(xdist + ydist + zdist)
```

Gambar 2.1.4. Fungsi EuclideanDist3 untuk melakukan kalkulasi jarak antara 2 titik dalam 3 dimensi dengan persamaan jarak euclidean.

5. Fungsi EuclideanDistGeneral

```
# 4. Persamaan Euclidean distance (Tergeneralisasi) [BONUS]
def EuclideanDistGeneral(point1, point2, dimation) :
    # Persamaan : sqrt((x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2 + ...)
    # Inisialisasi senarai hasil
    hasil = [0 for i in range (dimation)]

    # Melakukan pemrosesan per subdimensi dan kalkulasi hasil total
    result = 0
    for i in range (dimation) :
        hasil[i] = pow(point1[i] - point2[i], 2)
        result += hasil[i]

    # Finalisasi hasil
    return math.sqrt(result)
```

Gambar 2.1.5. Fungsi EuclideanDistGeneral untuk melakukan kalkulasi jarak antara 2 titik dalam dimensi R^n dengan n menyatakan jumlah dimensi menggunakan persamaan jarak euclidean.

6. Implementasi algoritma pengurutan *QuickSort*

```
# 5. Implementasi QuickSort
# 5.1. Partisi untuk performansi QuicSort
def Partition(listOfPoints, min, max, target):
    pivot = listOfPoints[max] # menentukan pivot
    i = min - 1 # penunjuk i

    # bandingkan nilai dengan pivot
    for j in range(min, max):
        if listOfPoints[j][target] <= pivot[target]:
            # kalau lebih kecil, pindah penunjuk i ke elemen selanjutnya
            i = i + 1
            # Prosedur swap
            temp = listOfPoints[i]
            listOfPoints[i] = listOfPoints[j]
            listOfPoints[j] = temp

    # jangan lupa swap yang terakhir
    temp = listOfPoints[i + 1]
    listOfPoints[i + 1] = listOfPoints[max]
    listOfPoints[max] = temp

    # kembalikan posisi partisi
    return i + 1
```

Gambar 2.1.6. Fungsi Partition untuk menentukan batas partisi dengan algoritma *QuickSort*.

```
# 5.2. Algoritma utama QuickSort
def QuickSort(listOfPoints, min, max, target):
    if min < max:
        # menentukan letak partisi berdasarkan prosedur sebelumnya
        pos = Partition(listOfPoints, min, max, target)
        # Jalankan rekursifitas untuk bagian kanan dan kiri pivot
        QuickSort(listOfPoints, min, pos - 1, target)
        QuickSort(listOfPoints, pos + 1, max, target)

    # Kembalikan hasil sort
    return listOfPoints
```

Gambar 2.1.7. Fungsi QuickSort sebagai implementasi utama algoritma *QuickSort*.

B. File *Divide and Conquer* (DnC.py)

Pada *file* ini diimplementasikan Algoritma *Divide and Conquer* yang sudah dibahas pada bagian sebelumnya. Berikut adalah tampilannya.

1. Impor modul eksternal

```
# File pengimplementasian Algoritma Divide and Conquer
# 0. Impor modul eksternal, termasuk file
import Point
import BruteForce
```

Gambar 2.2.1. Modul eksternal yang diimport untuk membantu jalannya program, termasuk *file* lain yang memungkinkan untuk diambil modulnya.

2. Fungsi Minimum

```
# 1. Definisi nilai minimum dari 2 nilai masukan
def Minimum(x, y):
    if (x > y) :
        return y
    else :
        return x
```

Gambar 2.2.2. Fungsi Minimum untuk mendefinisikan nilai minimum antara dua masukan.

3. Fungsi LineCenterClosest

```
# 2. Pencarian titik terdekat yang berada di tengah dengan batas toleransi delta
def LineCenterClosest(lineCenter, size, minimum, am, bm, dimention):
    # Pendefinisian nilai dari parameter, sebagai handler kalau nilainya sama
    min_dist = minimum
    min_p1 = am
    min_p2 = bm

    # Membuat lineCenter dengan melakukan pengurutan berdasarkan nilai koordinat [1]
    lineCenter = Point.QuickSort(lineCenter, 0, len(lineCenter) - 1, 1)

    # Melakukan pengujian jarak terdekat dari titik disekitar pusat dalam batas toleransi
    count = 0
    for i in range(size):
        for j in range(i+1, size):
            # Kalau ukurannya lebih besar dari delta, lewati
            if abs(lineCenter[j][1] - lineCenter[i][1]) >= min_dist/2:
                break
            else :
                # Kalau lebih kecil, update nilai terkecilnya dan titik yang berkoresponden
                titik = Point.EuclideanDistGeneral(lineCenter[i], lineCenter[j], dimention)
                count += 1
                if titik <= min_dist:
                    min_dist = titik
                    min_p1 = lineCenter[i]
                    min_p2 = lineCenter[j]

    # Pengembalian dua titik terdekat dalam batas lineCenter dan jaraknya
    return min_p1, min_p2, min_dist, count
```

Gambar 2.2.3. Fungsi LineCenterClosest untuk menentukan jarak dari 2 titik terkecil yang berada di sekitar area pusat wilayah partisi dengan toleransi delta sebesar minimum.

4. Fungsi ClosestPairPoint3

```
# 3. Implementasi rekursif cari kanan kiri (3 dimensi)
def ClosestPairPoint3(listOfPoints, numbers) :
    # Kondisi pemberhenti rekursifitas, saat jumlah elemen partisi sudah terbatas
    if numbers <= 3:
        a, b, dist, count = BruteForce.BruteForceDist(listOfPoints)
        # Mengembalikan 2 titik dan nilai jaraknya
        return a, b, dist, count

    # Sebelum melakukan pemrosesan, titik diurutkan menaik berdasarkan nilai koordinat [0]
    sorted = Point.QuickSort(listOfPoints, 0, len(listOfPoints) - 1, 0)

    # Ambil titik tengah partisi
    midPoint = numbers // 2

    # Mencari titik dengan jarak terdekat di kanan dan kiri lineCenter secara rekursif
    a1, b1, dist_left, count1 = ClosestPairPoint3(sorted[:midPoint], midPoint)
    a2, b2, dist_right, count2 = ClosestPairPoint3(sorted[midPoint:], numbers - midPoint)

    # Mengambil nilai minimum dari jarak terdekat bagian kanan dan kiri serta titiknya
    min = Minimum(dist_left, dist_right)
    if (min == dist_left) :
        am = a1
        bm = b1
    else :
        am = a2
        bm = b2
```

Gambar 2.2.4. Fungsi ClosestPairPoint3 untuk melakukan prosedur penghitungan jarak 2 titik terdekat dari masing-masing wilayah partisi secara rekursif untuk bidang 3 dimensi – bagian (1).

```
# Mendefinisikan lineCenters, mengisi elemen linecenter dengan batas nilai delta, yaitu min
lineCenter = []
for i in range(numbers):
    if abs(sorted[i][0] - sorted[midPoint][0]) < min:
        lineCenter.append(sorted[i])

# Mengambil nilai terkecil disekitar lineCenter
p1, p2, val, count3 = LineCenterClosest(lineCenter, len(lineCenter), min, am, bm, 3)

# Membandingkan nilai sekitar lineCenter dengan nilai minimum partisi dan ambil titiknya
min_pol = Minimum(min, val)
if (min_pol == min) :
    a_min = am
    b_min = bm
else :
    a_min = p1
    b_min = p2

count_total = count1 + count2 + count3

# mengembalikan hasil final, yaitu kedua titik dengan jarak terdekat dan jaraknya
return a_min, b_min, min_pol, count_total
```

Gambar 2.2.5. Fungsi ClosestPairPoint3 – bagian (2).

5. Fungsi ClosestPairPointGeneral

```
# 4. Implementasi rekursif cari kanan kiri (Tergeneralisasi) [BONUS]
def ClosestPairPointGeneral(listOfPoints, numbers, dimention) :
    # Kondisi pemberhenti rekursifitas, saat jumlah elemen partisi sudah terbatas
    if numbers <= 3:
        a, b, dist, count = BruteForce.BruteForceDistGeneral(listOfPoints, dimention)
        # Mengembalikan 2 titik dan nilai jaraknya
        return a, b, dist, count

    # Sebelum melakukan pemrosesan, titik diurutkan menaik berdasarkan nilai koordinat [0]
    sorted = Point.QuickSort(listOfPoints, 0, len(listOfPoints) - 1, 0)

    # Ambil titik tengah partisi
    midPoint = numbers // 2

    # Mencari titik dengan jarak terdekat di kanan dan kiri lineCenter secara rekursif
    a1, b1, dist_left, count1 = ClosestPairPointGeneral(sorted[:midPoint], midPoint, dimention)
    a2, b2, dist_right, count2 = ClosestPairPointGeneral(sorted[midPoint:], numbers - midPoint, dimention)

    # Mengambil nilai minimum dari jarak terdekat bagian kanan dan kiri serta titiknya
    min = Minimum(dist_left, dist_right)
    if (min == dist_left) :
        am = a1
        bm = b1
    else :
        am = a2
        bm = b2
```

Gambar 2.2.6. Fungsi ClosestPairPointGeneral untuk melakukan prosedur penghitungan jarak 2 titik terdekat dari masing-masing wilayah partisi secara rekursif untuk bidang R^n – bagian (1).

```
# Mendefinisikan lineCenters, mengisi elemen linecenter dengan batas nilai delta, yaitu min
lineCenter = []
if (dimention > 1):
    for i in range(numbers):
        if abs(sorted[i][0] - sorted[midPoint][0]) < min:
            lineCenter.append(sorted[i])

# Mengambil nilai terkecil disekitar lineCenter
p1, p2, val, count3 = LineCenterClosest(lineCenter, len(lineCenter), min, am, bm, dimention)

# Membandingkan nilai sekitar lineCenter dengan nilai minimum partisi dan ambil titiknya
min_pol = Minimum(min, val)
if (min_pol == min) :
    a_min = am
    b_min = bm
else :
    a_min = p1
    b_min = p2

count_total = count1 + count2 + count3

# mengembalikan hasil final, yaitu kedua titik dengan jarak terdekat dan jaraknya
return a_min, b_min, min_pol, count_total
```

Gambar 2.2.7. Fungsi ClosestPairPointGeneral – bagian (2).

C. File Brute Force (BruteForce.py)

Pada *file* ini, selain mengimplementasikan Algoritma *Divide and Conquer*, juga diimplementasikan sebuah algoritma pembandingan, yaitu Algoritma *Brute Force*. Berikut adalah tampilannya.

1. Impor modul eksternal

```
# File pengimplementasian Algoritma Brute Force
# 0. Impor modul eksternal, termasuk file
import Point
```

Gambar 2.3.1. Modul eksternal yang diimpor untuk membantu jalannya program, termasuk *file* lain yang memungkinkan untuk diambil modulnya.

2. Fungsi BruteForceDist

```
# 1. Implementasi Strategi Bruteforce untuk titik 3 dimensi
def BruteForceDist(listOfPoints) :
    compare = 0 # Inisiasi jumlah perbandingan yang dilakukan
    shortest = Point.EuclideanDist3(listOfPoints[0], listOfPoints[1])

    for i in range (len(listOfPoints)) :
        for j in range(i+1, len(listOfPoints)) :
            compare += 1
            if (Point.EuclideanDist3(listOfPoints[i], listOfPoints[j]) <= shortest):
                shortest = Point.EuclideanDist3(listOfPoints[i], listOfPoints[j])
                a = listOfPoints[i]
                b = listOfPoints[j]

    return a, b, shortest, compare
```

Gambar 2.3.2. Fungsi BruteForceDist untuk melakukan Algoritma *Brute Force* pada setiap pasangan titik pada bidang 3 dimensi.

3. Fungsi BruteForceDistGeneral

```
# 2. Implementasi Strategi Bruteforce untuk dimensi tergeneralisasi [BONUS]
def BruteForceDistGeneral(listOfPoints, dimation) :
    compare = 0 # Inisiasi jumlah perbandingan yang dilakukan
    shortest = Point.EuclideanDistGeneral(listOfPoints[0], listOfPoints[1], dimation)

    for i in range (len(listOfPoints)) :
        for j in range(i+1, len(listOfPoints)) :
            compare += 1
            if (Point.EuclideanDistGeneral(listOfPoints[i], listOfPoints[j], dimation) <= shortest):
                shortest = Point.EuclideanDistGeneral(listOfPoints[i], listOfPoints[j], dimation)
                a = listOfPoints[i]
                b = listOfPoints[j]

    return a, b, shortest, compare
```

Gambar 2.3.3. Fungsi BruteForceDistGeneral untuk melakukan Algoritma *Brute Force* pada setiap pasangan titik pada bidang R^n .

D. File Visualisasi (Visual.py)

Pada *file* ini, diimplementasikan hasil visualisasi titik pada bidang 3 dimensi dan beberapa prosedur tambahan untuk mencetak titik, baik pada *terminal* maupun pada *file*. Berikut adalah tampilannya.

1. Impor modul eksternal

```
# File pengimplementasian Visualisasi Grafik dan Pencetakan Layar
# 0. Impor modul eksternal
import matplotlib.pyplot as plt
```

Gambar 2.4.1. Modul eksternal yang diimpor untuk membantu jalannya program.

2. Prosedur VisualizeMinimum

```
# 1. Visualisasi titik dalam bidang 3 Dimensi - Perjelas Minimum [BONUS]
def VisualizeMinimum(listOfPoints, point1, point2) :
    fig = plt.figure(figsize=(11,7))
    ax = fig.add_subplot(111, projection='3d')

    for i in range (len(listOfPoints)) :
        x = listOfPoints[i][0]
        y = listOfPoints[i][1]
        z = listOfPoints[i][2]

        ax.scatter(x,y,z, color='black')

    ax.scatter(point1[0],point1[1],point1[2], color='red')
    ax.scatter(point2[0],point2[1],point2[2], color='red')

    plt.show()
```

Gambar 2.4.2. Prosedur VisualizeMinimum untuk melakukan visualisasi terhadap semua titik dan pasangan titik dengan jarak terdekat dalam bidang 3 dimensi.

3. Prosedur PrintPoint

```
# 2. Cetak titik dalam bentuk kurung di terminal
def printPoint(Point) :
    n = len(Point) - 1
    a = "("
    for i in range (len(Point) - 1) :
        a += str(Point[i])
        a += ", "
    a += str(Point[n])
    a += ")"
    print(a, end="")
```

Gambar 2.4.3. Prosedur PrintPoint untuk melakukan pencetakan titik dalam format yang telah ditentukan pada *terminal*.

4. Prosedur PrintPointFile

```
# 3. Cetak titik dalam bentuk kurung di file
def printPointFile(Point, f) :
    n = len(Point) - 1
    a = "("
    for i in range (len(Point) - 1) :
        a += str(Point[i])
        a += ", "
    a += str(Point[n])
    a += ")"
    print(a, file=f)
```

Gambar 2.4.4. Prosedur PrintPointFile untuk melakukan pencetakan titik dalam format yang telah ditentukan pada *file* jika jumlah titik yang diminta terlalu banyak.

5. Prosedur PrintPointMatrix

```
# 3. Cetak titik dalam bentuk kurung di file
def printPointFile(Point, f) :
    n = len(Point) - 1
    a = "("
    for i in range (len(Point) - 1) :
        a += str(Point[i])
        a += ", "
    a += str(Point[n])
    a += ")"
    print(a, file=f)
```

Gambar 2.4.5. Prosedur PrintPointMatrix untuk melakukan pencetakan sekumpulan titik yang telah dibangkitkan dalam format yang telah ditentukan pada *terminal*.

6. Prosedur PrintPointMatrixFile

```
# 5. Cetak titik dari matriks di file
def printPointMatrixFile(ListOfPoint, f) :
    long = len(ListOfPoint) - 1

    for i in range (long) :
        n = len(ListOfPoint[i]) - 1
        a = "("

        for j in range (len(ListOfPoint[i]) - 1) :
            a += str(ListOfPoint[i][j])
            a += ", "
        a += str(ListOfPoint[i][n])
        a += "), "

        print(a, file=f)

    printPointFile(ListOfPoint[long], f)
```

Gambar 2.4.6. Prosedur PrintPointMatrixFile untuk melakukan pencetakan sekumpulan titik yang telah dibangkitkan dalam format yang telah ditentukan pada *file* jika jumlah titik yang diminta terlalu banyak.

E. *File* Program Utama (Main.py)

Pada program utama diimplementasikan berbagai fungsi dan prosedur yang telah didefinisikan pada berbagai *file* yang sudah dijabarkan sebelumnya untuk membentuk fungsionalitas utama program. Berikut adalah tampilannya.

```
# File pengimplementasi Program Utama
# Impor modul eksternal, termasuk file
import os
import time
import Point
import DnC
import BruteForce
import Visual
```

Gambar 2.5.1. Program Utama - bagian (1).

Gambar 2.5.2. Program Utama - bagian (2).

```
# Tampilan pilihan menu
print("===== INPUT =====")
print("Data titik seperti apa yang kamu pilih ?")
print("1. Tiga dimensi")
print("2. N dimensi")
print("Masukkan pilihanmu [1/2]")
pilihan = int(input(">> "))

# Validasi masukan
while (pilihan != 1 and pilihan != 2):
    print("> Pilihan tidak valid, ulangi!")
    print("-----")
    print("Data titik seperti apa yang kamu pilih ?")
    print("1. Tiga dimensi")
    print("2. N dimensi")
    print("Masukkan pilihanmu [1/2]")
    pilihan = int(input(">> "))
```

Gambar 2.5.3. Program Utama - bagian (3).


```

# 3. Implementasi rekursif cari kanan kiri (3 dimensi)
def ClosestPairPoint3(listOfPoints, numbers) :
    # Kondisi pemberhenti rekursifitas, saat jumlah elemen partisi sudah terbatas
    if numbers <= 3:
        a, b, dist, count = BruteForce.BruteForceDist(listOfPoints)
        # Mengembalikan 2 titik dan nilai jaraknya
        return a, b, dist, count

    # Sebelum melakukan pemrosesan, titik diurutkan menaik menggunakan fungsi sortPoints
    sorted = Point.SortPoints(listOfPoints)

    # Ambil titik tengah partisi
    midPoint = numbers // 2

    # Mencari titik dengan jarak terdekat di kanan dan kiri lineCenter secara rekursif
    a1, b1, dist_left, count1 = ClosestPairPoint3(sorted[:midPoint], midPoint)
    a2, b2, dist_right, count2 = ClosestPairPoint3(sorted[midPoint:], numbers - midPoint)

    # Mengambil nilai minimum dari jarak terdekat bagian kanan dan kiri serta titiknya
    min = Minimum(dist_left, dist_right)
    if (min == dist_left) :
        am = a1
        bm = b1
    else :
        am = a2
        bm = b2

```

Gambar 2.5.4. Program Utama - bagian (4).

```

print("\n===== HASIL ALGORITMA =====")
start_bf = time.time()
pts1, pts2, shortest1, compare1 = BruteForce.BruteForceDist(pointMatrix)
finish_bf = time.time()
start_dc = time.time()
pts3, pts4, shortestc1n1, compare2 = DnC.ClosestPairPoint3(pointMatrix, titik)
finish_dc = time.time()
print("Algoritma BruteForce --")
print('Pasangan titik terdekat : ', end="")
Visual.printPoint(pts1)
print(", ", end="")
Visual.printPoint(pts2)
print(f"\nJarak : {shortest1}")
print("\nAlgoritma Divide and Conquer --")
print('Pasangan titik terdekat : ', end="")
Visual.printPoint(pts3)
print(", ", end="")
Visual.printPoint(pts4)
print(f"\nJarak : {shortestc1n1}")

print("\n===== STATISTIK =====")
print("Algoritma BruteForce --")
print(f'Jumlah perbandingan : {compare1}')
print(f'Waktu Eksekusi : {finish_bf - start_bf} sekon\n')
print("Algoritma Divide and Conquer --")
print(f'Jumlah perbandingan : {compare2}')
print(f'Waktu Eksekusi : {finish_dc - start_dc} sekon')

```

Gambar 2.5.5. Program Utama - bagian (5).

```

# Penampilan grafik
print("\n===== PENAMPILAN GRAFIK =====")
print("Apakah ingin menampilkan hasil ilustrasi titik ?")
print("Masukkan pilihanmu [Y/n]")
pilihan = input(">> ")

# Validasi masukan
while (pilihan != "Y" and pilihan != "y" and pilihan != "N" and pilihan != "n"):
    print("> Pilihan tidak valid, ulang!")
    print("-----")
    print("Apakah ingin menampilkan hasil ilustrasi titik ?")
    print("Masukkan pilihanmu [Y/n]")
    pilihan = input(">> ")

# Pemrosesan masukan
if (pilihan == "Y" or pilihan == "y"):
    print("\nPemrosesan sedang berlangsung...")
    Visual.VisualizeMinimum(pointMatrix, pts1, pts2)
    print(" ")
else :
    print(" ")

```

Gambar 2.5.6. Program Utama - bagian (6).

```

else :
    print("\n===== PASANGAN N DIMENSI =====")
    print("Masukkan jumlah dimensi")
    dimensi = int(input(">> "))
    print("Masukkan jumlah titik")
    titik = int(input(">> "))
    pointMatrix = Point.ListRandomPoint(dimensi, titik)

    print("\n===== PEMBANGKITAN TITIK ACAK =====")
    print("Daftar kumpulan titik")

    # Case handling untuk masukan diatas 50
    if titik <= 50 :
        Visual.printPointMatrix(pointMatrix)
        print("")
    else :
        print("Karena jumlah titik yang banyak, maka data titik akan disimpan pada Result.txt")
        with open('Result.txt', 'w') as f:
            print('Daftar titik yang dibangkitkan :\n', file=f)
            Visual.printPointMatrixFile(pointMatrix, f)

```

Gambar 2.5.7. Program Utama - bagian (7).

```

print("\n===== HASIL ALGORITMA =====")
start_bf = time.time()
pts1, pts2, shortest1, compare1 = BruteForce.BruteForceDistGeneral(pointMatrix, dimensi)
finish_bf = time.time()
start_dc = time.time()
pts3, pts4, shortestc1n1, compare2 = DnC.ClosestPairPointGeneral(pointMatrix, titik, dimensi)
finish_dc = time.time()
print("Algoritma BruteForce --")
print('Pasangan titik terdekat : ', end="")
Visual.printPoint(pts1)
print(", ", end= "")
Visual.printPoint(pts2)
print(f"\nJarak          : {shortest1}")
print("\nAlgoritma Divide and Conquer --")
print('Pasangan titik terdekat : ', end="")
Visual.printPoint(pts3)
print(", ", end= "")
Visual.printPoint(pts4)
print(f"\nJarak          : {shortestc1n1}")

print("\n===== STATISTIK =====")
print("Algoritma BruteForce --")
print(f'Jumlah perbandingan : {compare1}')
print(f'Waktu Eksekusi      : {finish_bf - start_bf} sekon\n')
print("Algoritma Divide and Conquer --")
print(f'Jumlah perbandingan : {compare2}')
print(f'Waktu Eksekusi      : {finish_dc - start_dc} sekon\n')

```

Gambar 2.5.8. Program Utama - bagian (8).


PROGRAM TESTING

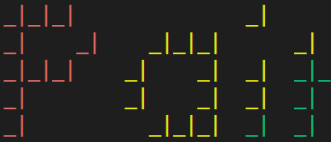
A. Tampilan Awal

Berikut adalah tampilan awal program yang telah dibuat.

```

= Tugas Kecil 2 Strategi Algoritma =





Points Detector -  

Divide and Conquer Algorithm

Made by :  

13221063 / 13221108

===== INPUT =====
Data titik seperti apa yang kamu pilih ?
1. Tiga dimensi
2. N dimensi
Masukkan pilihanmu [1/2]
>> 1

```

Gambar 3.1.1. Tampilan awal program dan tampilan awal pemilihan menu input.

```
===== INPUT =====
Data titik seperti apa yang kamu pilih ?
1. Tiga dimensi
2. N dimensi
Masukkan pilihanmu [1/2]
>> 3
> Pilihan tidak valid, ulangi!
-----
Data titik seperti apa yang kamu pilih ?
1. Tiga dimensi
2. N dimensi
Masukkan pilihanmu [1/2]
>> makan
> Pilihan tidak valid, ulangi!
-----
Data titik seperti apa yang kamu pilih ?
1. Tiga dimensi
2. N dimensi
Masukkan pilihanmu [1/2]
>> []
```

Gambar 3.1.2. Skema pengendalian nilai masukan.

B. Skema Pemrosesan Solusi

Berikut adalah tampilan setelah program selesai dieksekusi.

```
===== INPUT =====
Data titik seperti apa yang kamu pilih ?
1. Tiga dimensi
2. N dimensi
Masukkan pilihanmu [1/2]
>> 1

===== PASANGAN 3 DIMENSI =====
Masukkan jumlah titik
>> 8

===== PEMBANGKITAN TITIK ACAK =====
Daftar kumpulan titik
(798.67, -284.1, 529.75),
(-330.41, 900.81, -933.79),
(195.41, -200.68, -803.39),
(-792.14, -180.82, 989.04),
(-337.86, -227.95, 920.0),
(566.32, -664.26, 437.89),
(-548.56, 276.11, 901.15),
(-553.42, -403.07, -383.78)

===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (798.67, -284.1, 529.75), (566.32, -664.26, 437.89)
Jarak                    : 454.9136266369693

Algoritma Divide and Conquer --
Pasangan titik terdekat : (566.32, -664.26, 437.89), (798.67, -284.1, 529.75)
Jarak                    : 454.9136266369693

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 28
Waktu Eksekusi      : 0.0 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 11
Waktu Eksekusi      : 0.0 sekon
```

Gambar 3.2.1. Tampilan hasil dan proses eksekusi program untuk bidang 3 dimensi.

```

===== INPUT =====
Data titik seperti apa yang kamu pilih ?
1. Tiga dimensi
2. N dimensi
Masukkan pilihanmu [1/2]
>> 2

===== PASANGAN N DIMENSI =====
Masukkan jumlah dimensi
>> 5
Masukkan jumlah titik
>> 6

===== PEMBANGKITAN TITIK ACAK =====
Daftar kumpulan titik
(989.44, -981.48, -583.45, -612.14, 131.02),
(518.2, 757.83, -336.73, -877.22, 739.67),
(-511.15, -2.91, -534.0, -888.66, -223.76),
(-967.74, 954.88, -619.19, 225.31, 524.45),
(-174.07, -78.73, 120.33, 418.03, 241.52),
(-341.45, 959.88, -139.42, -737.34, -754.95)

===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (-511.15, -2.91, -534.0, -888.66, -223.76), (-341.45, 959.88, -139.42, -737.34, -754.95)
Jarak : 1190.1750329258296

Algoritma Divide and Conquer --
Pasangan titik terdekat : (-511.15, -2.91, -534.0, -888.66, -223.76), (-341.45, 959.88, -139.42, -737.34, -754.95)
Jarak : 1190.1750329258296

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 15
Waktu Eksekusi : 0.0010004043579101562 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 18
Waktu Eksekusi : 0.0 sekon

```

Gambar 3.2.2. Tampilan hasil dan proses eksekusi program untuk bidang R^n , dalam hal ini, 5 dimensi.

C. Skema Penampilan Visualisasi

Berikut adalah tampilan proses visualisasi untuk titik dalam bidang 3 dimensi.

```

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 28
Waktu Eksekusi : 0.0 sekon

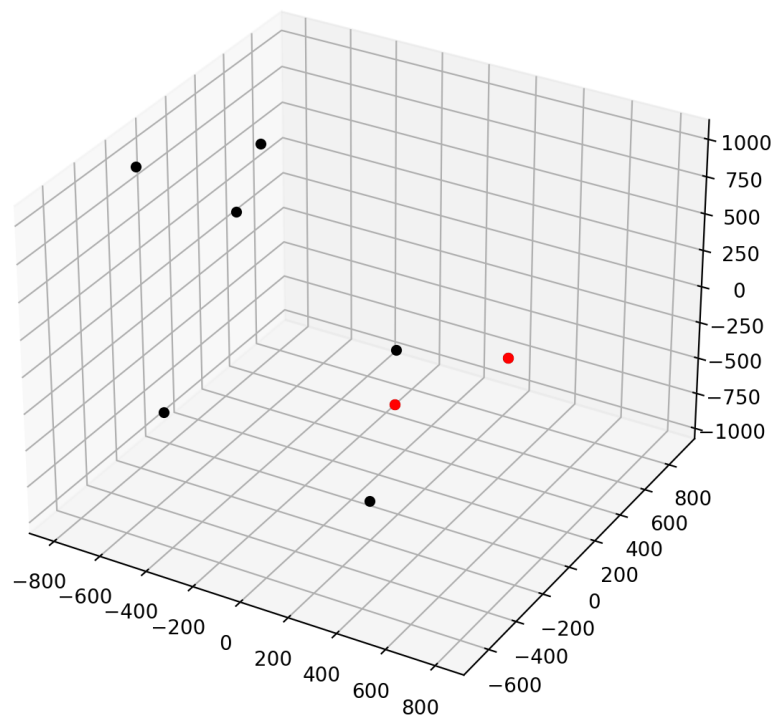
Algoritma Divide and Conquer --
Jumlah perbandingan : 11
Waktu Eksekusi : 0.0 sekon

===== PENAMPILAN GRAFIK =====
Apakah ingin menampilkan hasil ilustrasi titik ?
Masukkan pilihanmu [Y/n]
>> Y

Pemrosesan sedang berlangsung....
□

```

Gambar 3.3.1. Tampilan penawaran pengguna untuk melakukan visualisasi dan pengguna memilih hal tersebut.



Gambar 3.3.2. Hasil visualisasi menggunakan *library* matplotlib.

D. Test Case

1. Penyelesaian pada bidang 3 dimensi
 - a. Pasangan untuk $n = 16$

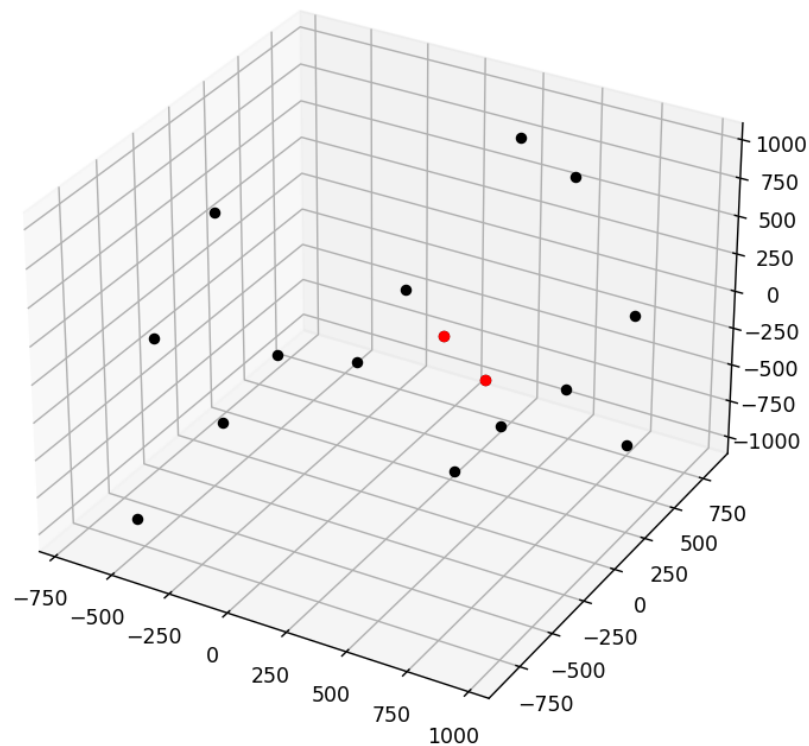
```
===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (823.05, -858.68, 966.34), (960.56, -813.44, 728.12)
Jarak                   : 278.7551364549181

Algoritma Divide and Conquer --
Pasangan titik terdekat : (823.05, -858.68, 966.34), (960.56, -813.44, 728.12)
Jarak                   : 278.7551364549181

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 120
Waktu Eksekusi      : 0.0 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 32
Waktu Eksekusi      : 0.0 sekon
```

Gambar 3.4.1.1. Hasil Test Case dengan 3 Dimensi dan 16 Titik



Gambar 3.4.1.2. Visualisasi Test Case dengan 3 Dimensi dan 16 Titik

b. Pasangan untuk $n = 64$

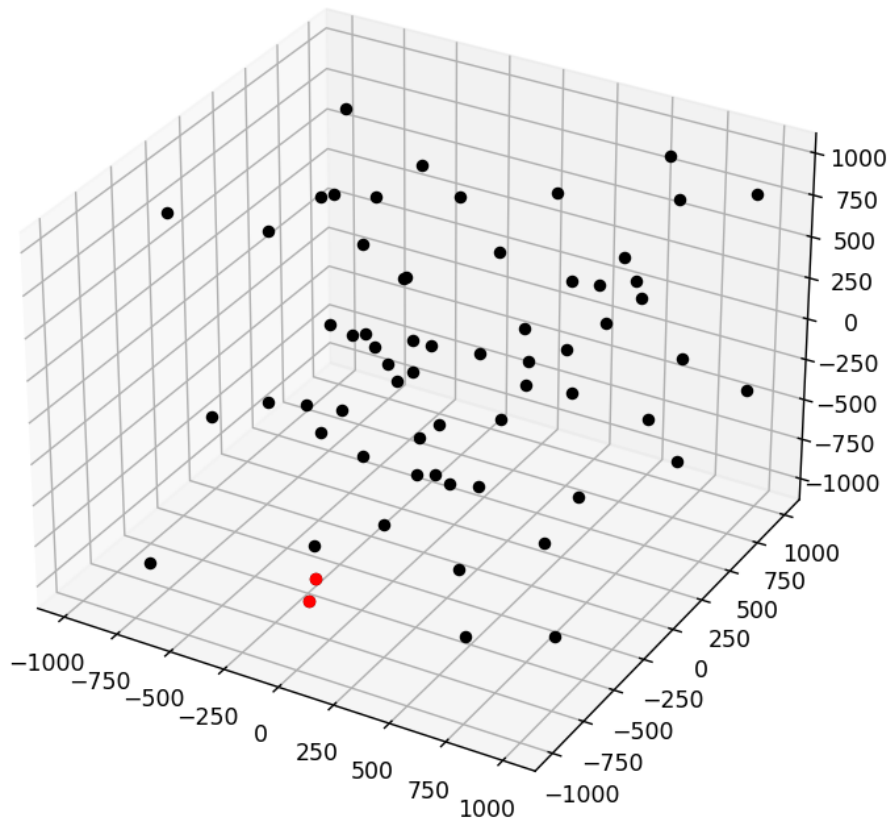
```
===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (-62.05, -795.91, -882.13), (-78.76, -721.65, -810.3)
Jarak                    : 104.65811292011722

Algoritma Divide and Conquer --
Pasangan titik terdekat : (-62.05, -795.91, -882.13), (-78.76, -721.65, -810.3)
Jarak                    : 104.65811292011722

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 2016
Waktu Eksekusi      : 0.002026081085205078 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 143
Waktu Eksekusi      : 0.0009937286376953125 sekon
```

Gambar 3.4.2.1. Hasil Test Case dengan 3 Dimensi dan 64 Titik



Gambar 3.4.2.2. Visualisasi Test Case dengan 3 Dimensi dan 64 Titik

c. Pasangan untuk $n = 128$

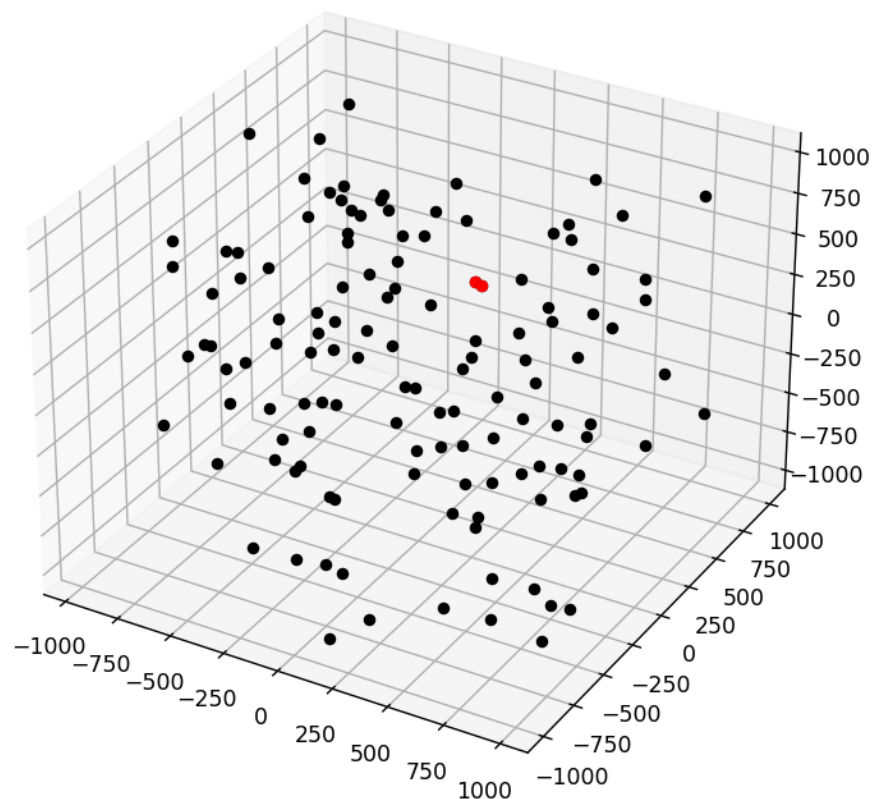
```
===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (9.17, 457.87, 221.54), (58.53, 413.85, 246.05)
Jarak                    : 70.5330426112471

Algoritma Divide and Conquer --
Pasangan titik terdekat : (58.53, 413.85, 246.05), (9.17, 457.87, 221.54)
Jarak                    : 70.5330426112471

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 8128
Waktu Eksekusi      : 0.006000518798828125 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 262
Waktu Eksekusi      : 0.004081249237060547 sekon
```

Gambar 3.4.3.1 Hasil Test Case dengan 3 Dimensi dan 128 Titik



Gambar 3.4.3.2 Visualisasi Test Case dengan 3 Dimensi dan 128 Titik

d. Pasangan untuk $n = 1000$

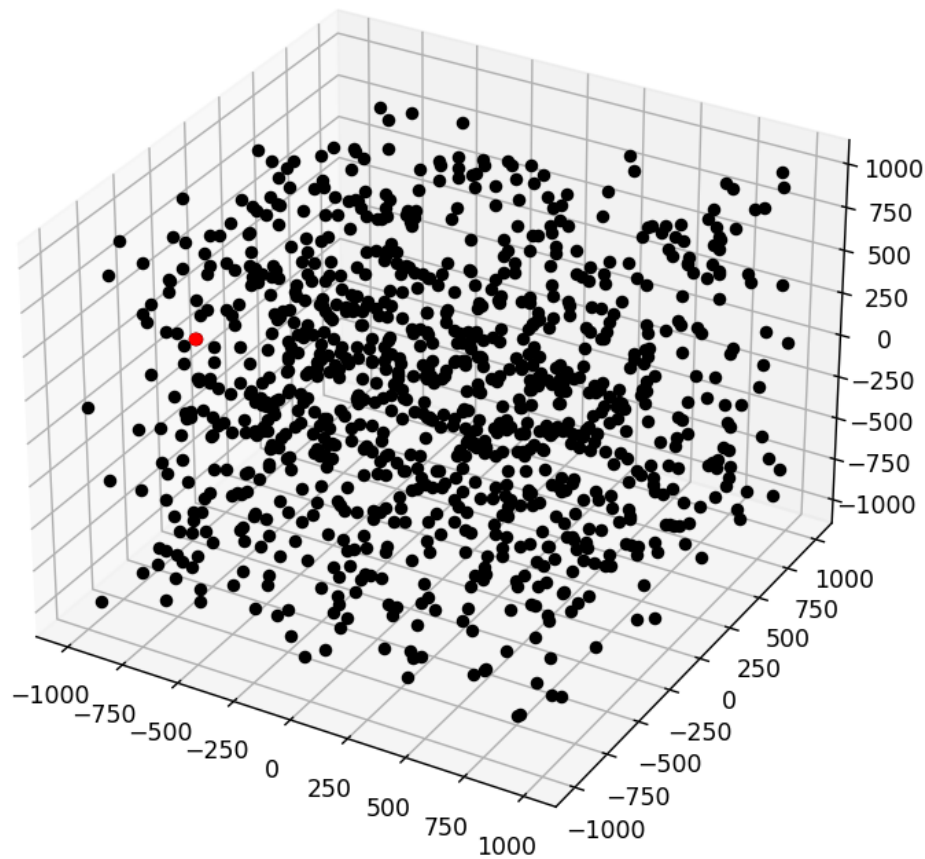
```
===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (-538.14, -194.69, -488.72), (-530.74, -184.36, -489.56)
Jarak                    : 12.734775223772083

Algoritma Divide and Conquer --
Pasangan titik terdekat : (-538.14, -194.69, -488.72), (-530.74, -184.36, -489.56)
Jarak                    : 12.734775223772083

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 499500
Waktu Eksekusi      : 0.34790468215942383 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 2737
Waktu Eksekusi      : 0.09222841262817383 sekon
```

Gambar 3.4.4.1 Hasil Test Case dengan 3 Dimensi dan 1000 Titik



Gambar 3.4.4.2 Visualisasi Test Case dengan 3 Dimensi dan 1000 Titik

2. Penyelesaian pada bidang R^n dengan $n > 3$
 - a. Pasangan untuk $n = 16$ dan dimensi = 6

```
===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (151.27, 516.61, 174.8, 579.85, 960.98, -80.73), (-59.83, 775.55, 212.18, 847.4, 807.51, -569.41)
Jarak : 668.5466205134836

Algoritma Divide and Conquer --
Pasangan titik terdekat : (151.27, 516.61, 174.8, 579.85, 960.98, -80.73), (-59.83, 775.55, 212.18, 847.4, 807.51, -569.41)
Jarak : 668.5466205134836

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 120
Waktu Eksekusi : 0.0 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 64
Waktu Eksekusi : 0.0 sekon
```

Gambar 3.4.5. Hasil Test Case dengan 6 Dimensi dan 16 Titik

- b. Pasangan untuk $n = 64$ dan dimensi = 4

```

===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (-457.75, 563.17, -431.99, 54.18), (-427.84, 509.56, -446.86, 216.27)
Jarak : 173.9624246784345

Algoritma Divide and Conquer --
Pasangan titik terdekat : (-457.75, 563.17, -431.99, 54.18), (-427.84, 509.56, -446.86, 216.27)
Jarak : 173.9624246784345

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 2016
Waktu Eksekusi : 0.003467798233032266 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 241
Waktu Eksekusi : 0.0009033679962158203 sekon

```

Gambar 3.4.6. Hasil Test Case dengan 4 Dimensi dan 64 Titik

- c. Pasangan untuk $n = 128$ dan dimensi = 10

```

===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (-487.05, 573.11, 651.72, 518.67, -905.67, -205.76, -218.86, -242.28, 66.6, -205.32), (-462.55, 620.36, 794.0, 805.46, -809.3, -220.08, -458.43, -389.8, 253.72, -469.91)
Jarak : 479.83419229979853

Algoritma Divide and Conquer --
Pasangan titik terdekat : (-487.05, 573.11, 651.72, 518.67, -905.67, -205.76, -218.86, -242.28, 66.6, -205.32), (-462.55, 620.36, 794.0, 805.46, -809.3, -220.08, -458.43, -389.8, 253.72, -469.91)
Jarak : 479.83419229979853

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 8128
Waktu Eksekusi : 0.029592514038085938 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 3193
Waktu Eksekusi : 0.014043569564819336 sekon

```

Gambar 3.4.7. Hasil Test Case dengan 10 Dimensi dan 128 Titik

- d. Pasangan untuk $n = 1000$ dan dimensi = 8

```

===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (866.61, 694.04, 673.17, 472.05, 879.56, 748.25, -939.46, -61.5), (791.39, 685.69, 789.47, 499.55, 774.85, 720.81, -893.11, -297.96)
Jarak : 299.64729716785365

Algoritma Divide and Conquer --
Pasangan titik terdekat : (791.39, 685.69, 789.47, 499.55, 774.85, 720.81, -893.11, -297.96), (866.61, 694.04, 673.17, 472.05, 879.56, 748.25, -939.46, -61.5)
Jarak : 299.64729716785365

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 499500
Waktu Eksekusi : 1.296494960784912 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 89658
Waktu Eksekusi : 0.3408498764038086 sekon

```

Gambar 3.4.8. Hasil Test Case dengan 8 Dimensi dan 1000 Titik

3. Penyelesaian pada bidang R^n dengan $n < 3$

- a. Pasangan untuk $n = 16$ dan dimensi = 1

```

===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (-941.76), (-942.76)
Jarak                    : 1.0

Algoritma Divide and Conquer --
Pasangan titik terdekat : (-942.76), (-941.76)
Jarak                    : 1.0

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 120
Waktu Eksekusi      : 0.0 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 9
Waktu Eksekusi      : 0.0 sekon

```

Gambar 3.4.9. Hasil Test Case dengan 1 Dimensi dan 16 Titik

- b. Pasangan untuk $n = 64$ dan dimensi = 2

```

===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (-316.76, -134.55), (-314.23, -135.18)
Jarak                    : 2.6072590972129825

Algoritma Divide and Conquer --
Pasangan titik terdekat : (-314.23, -135.18), (-316.76, -134.55)
Jarak                    : 2.6072590972129825

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 2016
Waktu Eksekusi      : 0.006442070007324219 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 50
Waktu Eksekusi      : 0.0 sekon

```

Gambar 3.4.10. Hasil Test Case dengan 2 Dimensi dan 64 Titik

- c. Pasangan untuk $n = 128$ dan dimensi = 1

```

===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (670.63), (670.58)
Jarak                   : 0.049999999999954525

Algoritma Divide and Conquer --
Pasangan titik terdekat : (670.58), (670.63)
Jarak                   : 0.049999999999954525

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 8128
Waktu Eksekusi      : 0.030996322631835938 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 73
Waktu Eksekusi      : 0.004998922348022461 sekon

```

Gambar 3.4.11. Hasil Test Case dengan 1 Dimensi dan 128 Titik

- d. Pasangan untuk $n = 1000$ dan dimensi = 2

```

===== HASIL ALGORITMA =====
Algoritma BruteForce --
Pasangan titik terdekat : (743.21, -937.05), (743.39, -937.0)
Jarak                   : 0.18681541692263368

Algoritma Divide and Conquer --
Pasangan titik terdekat : (743.21, -937.05), (743.39, -937.0)
Jarak                   : 0.18681541692263368

===== STATISTIK =====
Algoritma BruteForce --
Jumlah perbandingan : 499500
Waktu Eksekusi      : 0.5604476928710938 sekon

Algoritma Divide and Conquer --
Jumlah perbandingan : 798
Waktu Eksekusi      : 0.0798494815826416 sekon

```

Gambar 3.4.12. Hasil Test Case dengan 2 Dimensi dan 1000 Titik

BAB IV

LAMPIRAN

Berikut Repository Github:

https://github.com/mikeleo03/Tucil2_13521063_13521108

Status Penyelesaian : *Completed*

No.	Poin	Ya	Tidak
1.	Program berhasil dikompilasi tanpa kesalahan	✓	
2.	Program berhasil <i>running</i>	✓	
3.	Program dapat menerima masukan dan menuliskan luaran	✓	
4.	Luaran program sudah benar (solusi <i>closest pair</i> benar)	✓	
5.	Bonus 1 dikerjakan	✓	
6.	Bonus 2 dikerjakan	✓	