

# **Tugas Kecil 2 IF2211 Strategi Algoritma Semester II Tahun 2022/2023**

## **Penerapan Algoritma Divide and Conquer pada Closest Pair of Points Problem**

Disusun oleh:

Kelvin Rayhan    13521005  
Laila Bilbina    13521016



**STUDI TEKNIK  
INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO  
DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG 2022**

## 1. ALGORITMA

### 1. Algoritma *Brute Force*

Fungsi *brute\_force* adalah salah satu fungsi yang digunakan dalam fungsi *find\_closest\_pair*. Fungsi ini akan mencari sepasang titik terdekat dalam daftar *points* dengan cara brute force, yaitu dengan memeriksa setiap pasang titik dan mencari jarak terkecil di antara mereka.

- a. Fungsi *brute\_force* menerima satu parameter yaitu *points*, yaitu daftar titik-titik yang akan dicari sepasang titik terdekatnya.
- b. Variabel *min\_dist* diinisialisasi dengan nilai *infinity*, dan variabel *pair* belum diinisialisasi.
- c. Dalam loop pertama, fungsi *brute\_force* akan memeriksa setiap titik pada daftar *points*
- d. Dalam loop kedua, fungsi *brute\_force* akan memeriksa setiap titik pada daftar *points* yang belum diperiksa, dimulai dari titik berikutnya setelah titik yang sedang diperiksa.
- e. Fungsi *dist* dipanggil untuk menghitung jarak antara kedua titik, dan jarak tersebut disimpan dalam variabel *d*.
- f. Jika *d* lebih kecil dari *min\_dist*, maka *d* dijadikan nilai *min\_dist*, dan pasangan titik yang diperiksa saat ini dijadikan pasangan titik terdekat.
- g. Setelah loop selesai, fungsi *brute\_force* akan mengembalikan pasangan titik terdekat dan jaraknya.

### 2. Algoritma *Divide and Conquer*

- a. Pertama-tama, fungsi ini menerima sebuah argumen *points* yang merupakan daftar dari titik-titik pada bidang kartesius, di mana setiap titik direpresentasikan sebagai sepasang koordinat (*x*, *y*).
- b. Kemudian, fungsi *divide\_conquer* dipanggil dengan argumen *points\_x* dan *points\_y*. Fungsi *divide\_conquer* adalah implementasi dari algoritma *divide and conquer* untuk mencari sepasang titik terdekat. Algoritma ini bekerja dengan membagi setiap daftar titik menjadi dua bagian hingga hanya tersisa tiga titik atau kurang. Untuk kasus basis ketika daftar titik hanya berisi tiga titik atau kurang, fungsi akan menggunakan algoritma brute force untuk mencari sepasang titik terdekat. Jika tidak, maka daftar titik akan dibagi menjadi dua bagian dan fungsi *divide\_conquer* akan dipanggil secara rekursif untuk masing-masing bagian.
- c. Pada setiap panggilan rekursif, daftar titik akan diurutkan berdasarkan koordinat *x* dan *y* menggunakan fungsi *sorted*. Fungsi *sorted* akan mengembalikan daftar titik yang diurutkan berdasarkan koordinat *x* atau *y*, tergantung pada argumen yang diberikan kepadanya.
- d. Pada setiap panggilan rekursif, titik tengah dihitung sebagai titik tengah daftar titik *x* yang telah diurutkan. Titik-titik pada daftar titik *y* kemudian dibagi menjadi dua kelompok, yaitu titik-titik pada sebelah kiri dan kanan titik tengah.
- e. Setelah itu, sepasang titik terdekat pada masing-masing bagian dicari dengan menggunakan fungsi *divide\_conquer* secara rekursif. Hasil dari kedua bagian ini kemudian dicatat sebagai *pair\_left* dan *pair\_right* serta jarak antara sepasang titik terdekat pada masing-masing bagian kemudian dicatat sebagai *dist\_left* dan *dist\_right*.

- f. Selanjutnya, hasil pencarian dari kedua bagian di atas akan dibandingkan, sehingga akan didapatkan sepasang titik terdekat dan jarak antara sepasang titik terdekat dari seluruh titik pada daftar yang diberikan.
- g. Setelah itu, semua titik yang berada pada jalur "strip" (sebuah jalur yang terbentuk dari jarak antara titik tengah dan tepi terdekat), kemudian diurutkan berdasarkan koordinat y. Kemudian, setiap pasangan titik pada strip ini akan dicari jaraknya, dan bila ditemukan pasangan titik yang jaraknya lebih kecil dari jarak antara sepasang titik terdekat yang ditemukan pada langkah 6, maka pasangan titik tersebut akan menjadi sepasang titik terdekat yang baru.
- h. Akhirnya, fungsi `divide_conquer` mengembalikan sepasang titik terdekat yang ditemukan serta jarak antara sepasang titik terdekat tersebut.

CATATAN:

- Algoritma *Brute Force* dan *Divide and Conquer* dapat memberi hasil yang “terbalik” ( $\langle P1, P2 \rangle$  seharusnya ekuivalen dengan  $\langle P3, P4 \rangle$  namun bisa jadi seperti  $\langle P1, P2 \rangle$  dengan  $\langle P4, P3 \rangle$  dimana P4 dan P3 tertukar), hal ini merupakan *by design* dan tidak ditangani oleh program.
- 3D plotting hanya akan bisa dijalankan jika GNUPlot terinstal pada instalasi Windows (sesuai dengan requirement pada GitHub).
- Jika masukan plotting (y/yes or n/no) salah, maka program akan asumsi pilihan “no”. Ini merupakan *design choice*.
- Masukan n (jumlah point) tidak terbatas, namun minimal 2.
- Masukan dimensi dibataskan menjadi 1 sampai 10 (inklusif).

## 2. SOURCE CODE (C++)

Main.py

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import random
import time

c = 0

def dist(p1, p2):
    # menghitung jarak antara dua titik dalam ruang 3D
    return np.sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2 + (p1[2]-p2[2])**2)

def brute_force(points):
    # pencarian sepasang titik terdekat dengan algoritma brute force
    min_dist = float('inf')
    for i in range(len(points)):
        for j in range(i+1, len(points)):
            d = dist(points[i], points[j])
            if d < min_dist:
                min_dist = d
                pair = (points[i], points[j])
    return pair, min_dist

def find_closest_pair(points):
    global c
    # pencarian sepasang titik terdekat dengan algoritma divide and conquer
    def divide_conquer(points_x, points_y):
        n = len(points_x)
        global c
        if n <= 3:
            return brute_force(points_x)
        else:
            mid = n // 2
            mid_point = points_x[mid][0]
            points_y_left = [p for p in points_y if p[0] < mid_point]
            points_y_right = [p for p in points_y if p[0] >= mid_point]
            pair_left, dist_left = divide_conquer(points_x[:mid], points_y_left)
            pair_right, dist_right = divide_conquer(points_x[mid:], points_y_right)
            if dist_left < dist_right:
                closest_pair = pair_left
                closest_dist = dist_left
            else:
                closest_pair = pair_right
                closest_dist = dist_right
            strip_points = [p for p in points_y if abs(p[0] - mid_point) < closest_dist]
            strip_n = len(strip_points)
            for i in range(strip_n):
                j = i + 1
                while j < strip_n and strip_points[j][1] - strip_points[i][1] < closest_dist:
                    d = dist(strip_points[i], strip_points[j])
                    c += 1
                    if d < closest_dist:
                        closest_dist = d
                        closest_pair = (strip_points[i], strip_points[j])
                    j += 1
            return closest_pair, closest_dist

    points = sorted(points, key=lambda x: x[0])
    points_y = sorted(points, key=lambda x: x[1])
    return divide_conquer(points, points_y)[0]

# input
def tigaDimensi():
    print("")
    print("")
    print("")
    n = int(input("Masukkan jumlah titik: "))
    points = np.array([[random.randint(-1000, 1000), random.randint(-1000, 1000), random.randint(-1000, 1000)] for i in range(n)])
```

```

# cari sepasang titik terdekat dengan algoritma brute force
start_time = time.time()
brute_pair, brute_dist = brute_force(points)
end_time = time.time()
banyakOPBrute = len(points)*(len(points)-1)/2

print("")
print("BRUTE FORCE")
print("Brute Force Pair : ", brute_pair)
print("Banyaknya operasi perhitungan :", banyakOPBrute)
print("Jarak : ", brute_dist)
print("Waktu eksekusi : ", end_time - start_time, "detik")

# cari sepasang titik terdekat dengan algoritma divide and conquer
start_time = time.time()
divcon_pair = find_closest_pair(points)
divcon_dist = dist(divcon_pair[0], divcon_pair[1])
end_time = time.time()

print("")
print("")
print("DIVIDE AND CONQUER")
print("Divide and Conquer Pair : ", divcon_pair)
print("Banyaknya operasi perhitungan :", c)
print("Jarak : ", divcon_dist)
print("Waktu eksekusi : ", end_time - start_time, " detik")

# BONUS 1
# plot semua titik dalam bidang 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(points[:,0], points[:,1], points[:,2], c='g', marker='o')

# plot sepasang titik terdekat dengan algoritma brute force
ax.plot([brute_pair[0][0], brute_pair[1][0]], [brute_pair[0][1], brute_pair[1][1]], [brute_pair[0][2], brute_pair[1][2]], c='r')

# plot sepasang titik terdekat dengan algoritma divide and conquer
ax.plot([divcon_pair[0][0], divcon_pair[1][0]], [divcon_pair[0][1], divcon_pair[1][1]], [divcon_pair[0][2], divcon_pair[1][2]], c='r')

plt.show()

def nDimensi():
    # BONUS 2
    n = int(input("Masukkan jumlah titik : "))
    dim = int(input("Masukkan dimensi : "))
    points = np.array([random.randint(-1000, 1000) for i in range(dim) for j in range(n)])

    # cari sepasang titik terdekat dengan algoritma brute force
    start_time = time.time()
    brute_pair, brute_dist = brute_force(points)
    end_time = time.time()
    banyakOPBrute = len(points)*(len(points)-1)/2

    print("")
    print("BRUTE FORCE")
    print("Brute Force Pair : ", brute_pair)
    print("Banyaknya operasi perhitungan :", banyakOPBrute)
    print("Jarak : ", brute_dist)
    print("Waktu eksekusi : ", end_time - start_time, "detik")

    start_time = time.time()
    divcon_pair = find_closest_pair(points)
    divcon_dist = dist(divcon_pair[0], divcon_pair[1])
    end_time = time.time()

    print("")
    print("DIVIDE AND CONQUER")
    print("Divide and Conquer Pair : ")
    for i in range(n):
        for j in range(i+1, n):
            if divcon_dist == dist(points[i], points[j]):

```

```

        print("(", points[i],") dan (", points[j],")")
        break
    else:
        continue
    break
print("Jarak: ", divcon_dist)
print("Banyaknya operasi perhitungan: ", c)
print("Waktu eksekusi: ", end_time - start_time, "detik")

# BONUS 1
# plot semua titik dalam bidang 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(points[:,0], points[:,1], points[:,2], c='g', marker='o')

# plot sepasang titik terdekat dengan algoritma brute force
ax.plot([brute_pair[0][0], brute_pair[1][0]], [brute_pair[0][1], brute_pair[1][1]], [brute_pair[0][2], brute_pair[1][2]], c='r')

# plot sepasang titik terdekat dengan algoritma divide and conquer
ax.plot([divcon_pair[0][0], divcon_pair[1][0]], [divcon_pair[0][1], divcon_pair[1][1]], [divcon_pair[0][2], divcon_pair[1][2]], c='r')

plt.show()

def main():
    while True:
        print("=====")
        print("Program Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer")
        print("=====")
        print("1. 3 Dimensi")
        print("2. n Dimensi")
        print("3. Keluar")
        pilihan = int(input("Masukkan pilihan: "))
        if pilihan == 1:
            tigaDimensi()
        elif pilihan == 2:

```

```

        elif pilihan == 2:
            nDimensi()
        elif pilihan == 3:
            print("Terima kasih")
            break
        else:
            print("Pilihan tidak valid")

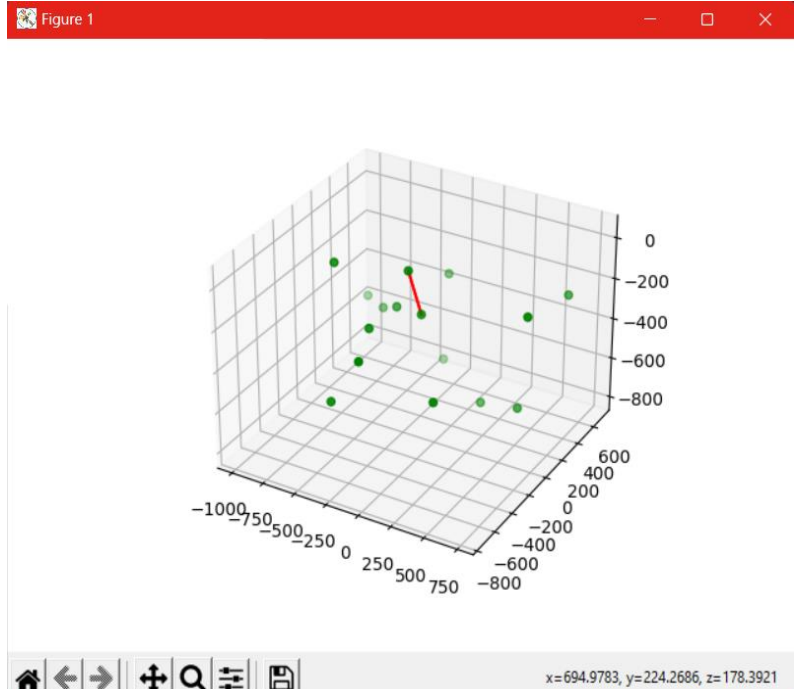
main()

```

### 3. CONTOH MASUKAN DAN LUARAN

Semua contoh dijalankan pada komputer dengan processor Intel Core i5-8250U, Quad-Core 1.6 GHz – 3.4 GHz, 6MB Cache

Dimensi 3

Test case (ukuran N)	Screenshot
16	<div><pre>===== Program Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer ===== 1. 3 Dimensi 2. n Dimensi 3. Keluar Masukkan pilihan: 1  Masukkan jumlah titik: 16  BRUTE FORCE Brute Force Pair : (array([ -86, -167,  30]), array([ -5, -140, -175])) Banyaknya operasi perhitungan : 120.0 Jarak : 222.06980884397592 Waktu eksekusi : 0.0 detik  DIVIDE AND CONQUER Divide and Conquer Pair : (array([ -86, -167,  30]), array([ -5, -140, -175])) Banyaknya operasi perhitungan : 18 Jarak : 222.06980884397592 Waktu eksekusi : 0.0008411407470703125 detik =====</pre></div> <div></div>

64

```

=====
Program Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer
=====
1. 3 Dimensi
2. n Dimensi
3. Keluar
Masukkan pilihan: 1

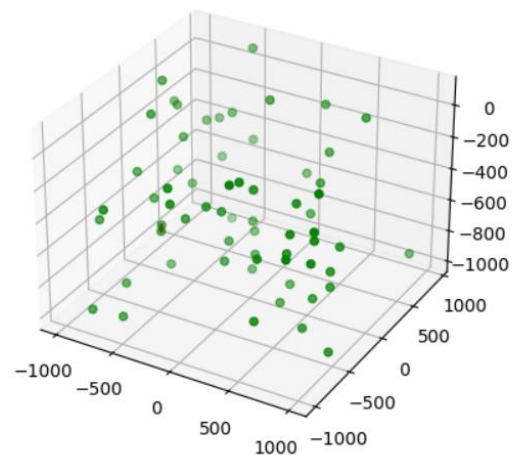
Masukkan jumlah titik: 64

BRUTE FORCE
Brute Force Pair : (array([-580, -321, -599]), array([-556, -364, -617]))
Banyaknya operasi perhitungan : 2016.0
Jarak : 52.43090691567332
Waktu eksekusi : 0.00766444206237793 detik

DIVIDE AND CONQUER
Divide and Conquer Pair : (array([-556, -364, -617]), array([-580, -321, -599]))
Banyaknya operasi perhitungan : 122
Jarak : 52.43090691567332
Waktu eksekusi : 0.0039975643157958984 detik
=====

```

Figure 1



x=1134.8295, y=1037.2826, z=-538.0240

128

```

=====
Program Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer
=====
1. 3 Dimensi
2. n Dimensi
3. Keluar
Masukkan pilihan: 1

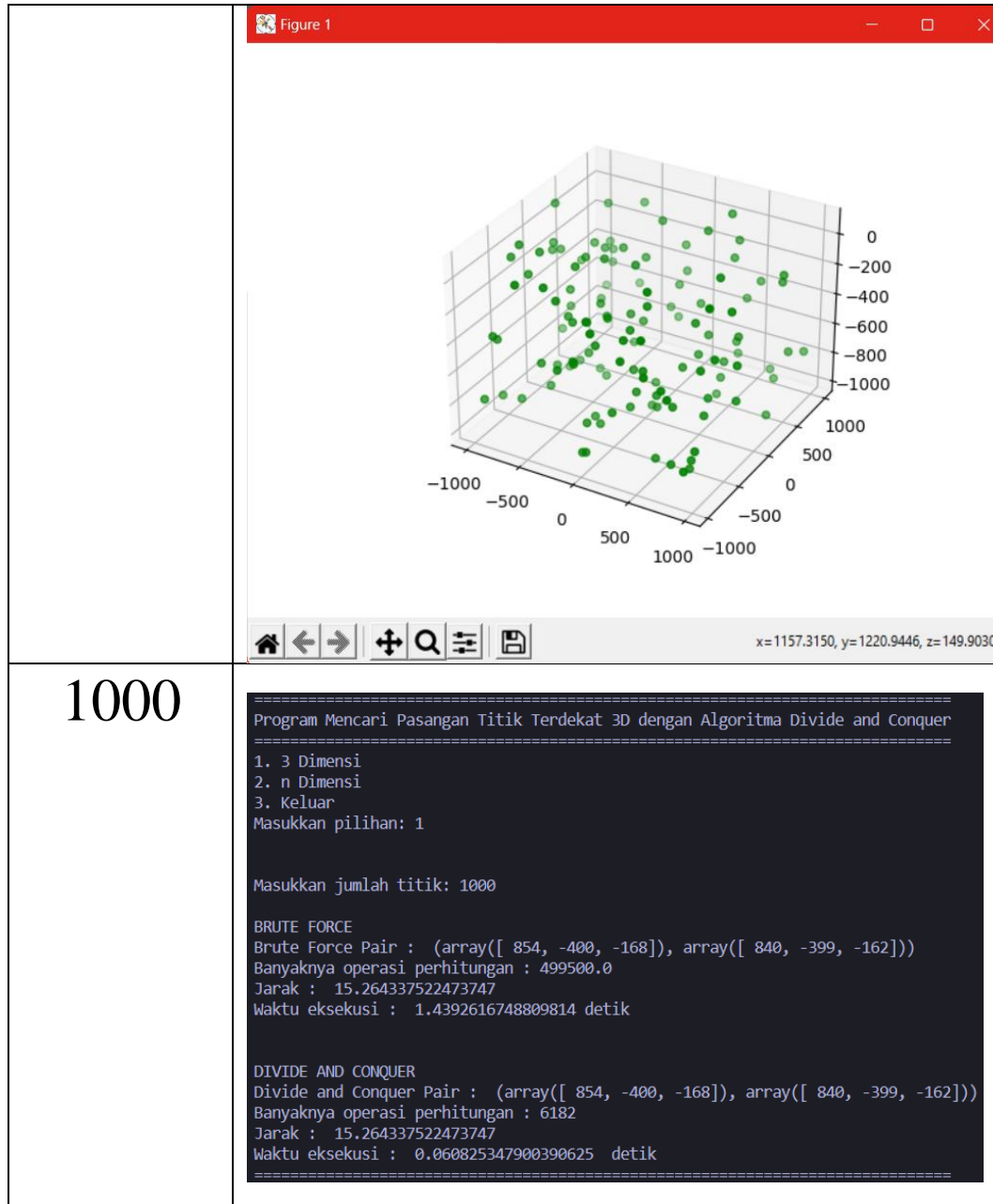
Masukkan jumlah titik: 128

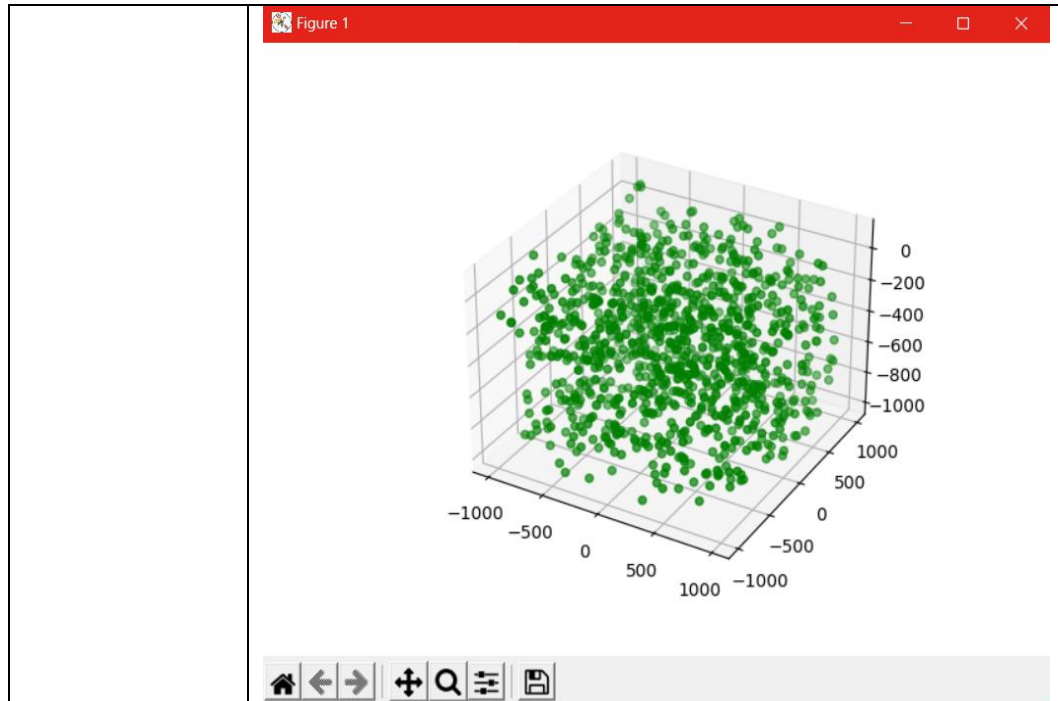
BRUTE FORCE
Brute Force Pair : (array([-101, -934, -351]), array([-116, -898, -384]))
Banyaknya operasi perhitungan : 8128.0
Jarak : 51.088159097779204
Waktu eksekusi : 0.04012751579284668 detik

DIVIDE AND CONQUER
Divide and Conquer Pair : (array([-101, -934, -351]), array([-116, -898, -384]))
Banyaknya operasi perhitungan : 3467
Jarak : 51.088159097779204
Waktu eksekusi : 0.004381418228149414 detik
=====

```







Dimensi  $n > 3$  (contoh yang dipakai adalah dimensi 10)

Test case (ukuran N)	Screenshot
16	<pre> Program Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer ===== 1. 3 Dimensi 2. n Dimensi 3. Keluar Masukkan pilihan: 2 Masukkan jumlah titik : 16 Masukkan dimensi : 10  BRUTE FORCE Brute Force Pair : (array([ 110,  813,   66, -754,  569, -589,  971,  293, -946, -524]), array([ 128,  660,   41,  445,   611, -179, -35, -880,   24,  122])) Banyaknya operasi perhitungan : 120.0 Jarak : 156.07049689162906 Waktu eksekusi : 0.0 detik  DIVIDE AND CONQUER Divide and Conquer Pair : ( [ 110  813   66 -754  569 -589  971  293 -946 -524] ) dan ( [ 128  660   41  445  611 -179 -35 -880   24  122] ) Jarak: 156.07049689162906 Banyaknya operasi perhitungan: 6196 Waktu eksekusi: 0.0 detik ===== </pre>

64	<pre> ===== Program Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer ===== 1. 3 Dimensi 2. n Dimensi 3. Keluar Masukkan pilihan: 2 Masukkan jumlah titik : 64 Masukkan dimensi : 10  BRUTE FORCE Brute Force Pair : (array([-58, -536, 626, -260, -927, 74, -720, -79, -816, 837]), array([-164, -543, 565, 321, 386, 809, -180, -663, 239, -422])) Banyaknya operasi perhitungan : 2016.0 Jarak : 122.49897958758677 Waktu eksekusi : 0.012346267700195312 detik  DIVIDE AND CONQUER Divide and Conquer Pair : ( [-58 -536 626 -260 -927 74 -720 -79 -816 837] ) dan ( [-164 -543 565 321 386 809 -180 -663 239 -422] ) Jarak: 122.49897958758677 Banyaknya operasi perhitungan: 6364 Waktu eksekusi: 0.004095315933227539 detik ===== </pre>
128	<pre> ===== Program Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer ===== 1. 3 Dimensi 2. n Dimensi 3. Keluar Masukkan pilihan: 2 Masukkan jumlah titik : 128 Masukkan dimensi : 10  BRUTE FORCE Brute Force Pair : (array([-42, -522, -85, 495, 149, 65, -999, 682, -817, -468], array([-41, -552, -64, -366, -731, 71, 506, 117, -505, -578])) Banyaknya operasi perhitungan : 8128.0 Jarak : 36.6333181680284 Waktu eksekusi : 0.028217792510986328 detik  DIVIDE AND CONQUER Divide and Conquer Pair : ( [-42 -522 -85 495 149 65 -999 682 -817 -468] ) dan ( [-41 -552 -64 -366 -731 71 506 117 -505 -578] ) Jarak: 36.6333181680284 Banyaknya operasi perhitungan: 6679 Waktu eksekusi: 0.012434959411621094 detik ===== </pre>
1000	<pre> ===== Program Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer ===== 1. 3 Dimensi 2. n Dimensi 3. Keluar Masukkan pilihan: 2 Masukkan jumlah titik : 1000 Masukkan dimensi : 10  BRUTE FORCE Brute Force Pair : (array([ 404, 684, -787, 207, 804, -661, 355, -194, -815, 671]), array([ 412, 695, -773, 491, -927, 284, -219, 298, -320, 672])) Banyaknya operasi perhitungan : 499500.0 Jarak : 19.519221295943137 Waktu eksekusi : 1.6606411933898926 detik  DIVIDE AND CONQUER Divide and Conquer Pair : ( [ 404 684 -787 207 804 -661 355 -194 -815 671] ) dan ( [ 412 695 -773 491 -927 284 -219 298 -320 672] ) Jarak: 19.519221295943137 Banyaknya operasi perhitungan: 10747 Waktu eksekusi: 0.05507016181945801 detik ===== </pre>

#### 4. LINK GITHUB

Link Repo GitHub: [https://github.com/Lailabkn/Tucil2\\_13521005\\_13521016.git](https://github.com/Lailabkn/Tucil2_13521005_13521016.git)

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	✓	