

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

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



Disusun oleh:  
Ilham Akbar      13521068

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

## 1. ALGORITMA

### 1. Algoritma *Brute Force*

Program di atas merupakan implementasi dari algoritma brute-force untuk mencari pasangan titik terdekat dalam suatu himpunan titik.

- a. Program menerima input berupa daftar titik-titik yang akan dicari pasangan titik terdekatnya.
- b. Variabel *n* diinisialisasi dengan panjang daftar titik, yaitu jumlah titik pada daftar.
- c. Variabel *distance\_terdekat* diinisialisasi dengan nilai tak hingga positif.
- d. Variabel *pasangan\_terdekat* diinisialisasi dengan nilai None.
- e. Variabel *count* diinisialisasi dengan nilai 0, yang akan digunakan untuk menghitung jumlah perhitungan jarak yang dilakukan oleh program.
- f. Program akan melakukan iterasi dua kali menggunakan loop for, dimana loop pertama akan mengambil titik *i* pada indeks 0 hingga *n*-2, dan loop kedua akan mengambil titik *j* pada indeks *i*+1 hingga *n*-1.
- g. Pada setiap iterasi loop, program akan menghitung jarak Euclidean antara titik *i* dan *j* menggunakan fungsi *jarak\_euclidean*, dan hasilnya disimpan pada variabel *distance*.
- h. Jumlah perhitungan jarak yang dilakukan oleh program (variabel *count*) akan ditambahkan 1.
- i. Jika jarak antara titik *i* dan *j* lebih kecil daripada *distance\_terdekat*, maka *distance\_terdekat* akan diupdate menjadi jarak tersebut dan *pasangan\_terdekat* akan diupdate menjadi pasangan titik (*i*, *j*).
- j. Setelah semua pasangan titik telah diiterasi, program akan mengembalikan nilai *pasangan\_terdekat*, *distance\_terdekat*, dan *count* sebagai output.

### 2. Algoritma *Divide and Conquer*

Program di atas merupakan implementasi dari algoritma Divide and Conquer untuk mencari pasangan titik terdekat dalam suatu himpunan titik. Algoritma Divide and Conquer bekerja dengan membagi himpunan titik menjadi dua sub-himpunan, memecahkan sub-himpunan secara rekursif, dan menggabungkan hasilnya untuk mendapatkan solusi akhir.

- a. Program menerima input berupa daftar titik-titik yang akan dicari pasangan titik terdekatnya.
- b. Pada awalnya, program akan memanggil fungsi *dnc* dengan parameter *points\_sorted*, yaitu daftar titik-titik yang telah diurutkan berdasarkan koordinat x-nya.
- c. Fungsi *dnc* akan menghitung jumlah titik pada daftar dan memeriksa apakah jumlahnya kurang dari atau sama dengan 3. Jika ya, maka fungsi *dnc* akan memanggil fungsi *brute\_force* untuk mencari pasangan titik terdekat dalam sub-himpunan tersebut.
- d. Jika jumlah titik pada daftar lebih dari 3, fungsi *dnc* akan membagi daftar titik menjadi dua sub-himpunan, yaitu titik-titik pada bagian kiri dan bagian kanan, dan memecahkan masing-masing sub-himpunan secara rekursif dengan memanggil fungsi *dnc*.
- e. Setelah mendapatkan pasangan titik terdekat dan jaraknya pada sub-himpunan kiri dan kanan, program akan menggabungkan hasilnya dengan mengambil pasangan titik terdekat yang memiliki jarak terdekat dari kedua

sub-himpunan.

- f. Program akan membangun strip yang terdiri dari titik-titik yang berada di sekitar garis tengah dan memiliki jarak Euclidean kurang dari jarak terdekat yang ditemukan pada sub-himpunan kiri dan kanan.
- g. Program akan melakukan perulangan untuk setiap pasangan titik pada strip, dan memeriksa apakah jarak antara kedua titik tersebut lebih kecil daripada jarak terdekat yang ditemukan sebelumnya. Jika ya, maka program akan memperbarui pasangan titik terdekat dan jarak terdekatnya.
- h. Setelah semua pasangan titik pada strip telah diiterasi, program akan mengembalikan nilai pasangan titik terdekat, jarak terdekat, dan jumlah perhitungan jarak yang dilakukan oleh program.

## 2. SOURCE CODE (C++)

main.py

```
main.py > main
1 import tucil2
2 import bonus2
3
4 def main():
5     while True:
6         print("=====")
7         print("----- Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer -----")
8         print("=====")
9         print("1. Dimensi Tiga")
10        print("2. Dimensi Banyak")
11        print("3. Keluar")
12        pilihan = int(input("Masukan Pilihan = "))
13        if pilihan == 1:
14            tucil2.dimensi_tiga()
15        elif pilihan == 2:
16            bonus2.Rn()
17        elif pilihan == 3:
18            print("Byee\n")
19            break
20        else:
21            print("Pilihan yang benar!!!\n")
22
23    main()
```

bonus2.py

```
1 import matplotlib.pyplot as plt
2 import random
3 import numpy as np
4 import time
5 import tucil2
6
7
8 def Rn():
9     # BONUS 2
10    n = int(input("Masukkan jumlah titik : "))
11    dimensi = int(input("Masukkan dimensi : "))
12    points = np.array([[random.randint(-1000, 1000) for i in range(dimensi)] for j in range(n)])
13
14    # cari sepasang titik terdekat dengan algoritma brute force
15    start_time = time.time()
16    brute_result, brute_distance, brute_count = tucil2.brute_force(points)
17    brute_time = time.time() - start_time
18    print("Hasil Brute Force:")
19    print("Sepasang titik terdekat:", brute_result)
20    print("Jarak terdekat:", brute_distance)
21    print("Banyaknya operasi perhitungan:", brute_count)
22    print("Waktu riil (detik):", brute_time)
23
24
25 # Algoritma divide and conquer
26 start_time = time.time()
27 dnc_result, dnc_distance, dnc_count = tucil2.divide_and_conquer(points)
28 dnc_time = time.time() - start_time
29 print("\nHasil Divide and Conquer:")
30 print("Sepasang titik terdekat:", dnc_result)
31 print("Jarak terdekat:", dnc_distance)
32 print("Banyaknya operasi perhitungan:", dnc_count)
33 print("Waktu riil (detik):", dnc_time)
34 print("\n")
35
36 plt.show()
```

## tucil2.py

```

1 import random
2 import math
3 import time
4 import matplotlib.pyplot as plt
5 from mpl_toolkits.mplot3d import Axes3D
6
7 # Fungsi untuk menghitung jarak antara dua titik
8 def jarak_euclidean(x, y):
9     return math.sqrt(sum([(x[i] - y[i]) ** 2 for i in range(len(x))]))
10
11 # Fungsi untuk mencari sepasang titik terdekat dengan algoritma brute force
12 def brute_force(points):
13     n = len(points)
14     distance_terdekat = float('inf')
15     pasangan_terdekat = None
16     count = 0
17
18     for i in range(n):
19         for j in range(i+1, n):
20             distance = jarak_euclidean(points[i], points[j])
21             count += 1
22             if distance < distance_terdekat:
23                 distance_terdekat = distance
24                 pasangan_terdekat = (points[i], points[j])
25
26     return pasangan_terdekat, distance_terdekat, count
27
28 # Fungsi untuk mencari sepasang titik terdekat dengan algoritma divide and conquer
29 def divide_and_conquer(points):
30     def dnc(points_sorted):
31         n = len(points_sorted)
32
33         if n <= 3:
34             return brute_force(points_sorted)
35
36         mid = n // 2
37         l_pair, l_distance, l_count = dnc(points_sorted[:mid])
38         r_pair, r_distance, r_count = dnc(points_sorted[mid:])
39         pasangan_terdekat = l_pair
40         distance_terdekat = l_distance
41         count = l_count + r_count
42
43         if r_distance < l_distance:
44             pasangan_terdekat = r_pair
45             distance_terdekat = r_distance
46
47         strip = []
48         for i in range(n):
49             if abs(points_sorted[i][0] - points_sorted[mid][0]) < distance_terdekat:
50                 strip.append(points_sorted[i])
51
52         for i in range(len(strip)):
53             for j in range(i+1, min(len(strip), i+7)):
54                 distance = jarak_euclidean(strip[i], strip[j])
55                 count += 1
56                 if distance < distance_terdekat:
57                     distance_terdekat = distance
58                     pasangan_terdekat = (strip[i], strip[j])
59
60     return pasangan_terdekat, distance_terdekat, count
61
62 points_sorted = sorted(points, key=lambda p: p[0])
63 return dnc(points_sorted)
64

```

```

tucil2.py • dimensi tiga
28 # Fungsi untuk mencari sepasang titik terdekat dengan algoritma divide and conquer
29 def divide_and_conquer(points):
30     def dnc(points_sorted):
31         n = len(points_sorted)
32
33         if n <= 3:
34             return brute_force(points_sorted)
35
36         mid = n // 2
37         l_pair, l_distance, l_count = dnc(points_sorted[:mid])
38         r_pair, r_distance, r_count = dnc(points_sorted[mid:])
39         pasangan_terdekat = l_pair
40         distance_terdekat = l_distance
41         count = l_count + r_count
42
43         if r_distance < l_distance:
44             pasangan_terdekat = r_pair
45             distance_terdekat = r_distance
46
47         strip = []
48         for i in range(n):
49             if abs(points_sorted[i][0] - points_sorted[mid][0]) < distance_terdekat:
50                 strip.append(points_sorted[i])
51
52         for i in range(len(strip)):
53             for j in range(i+1, min(len(strip), i+7)):
54                 distance = jarak_euclidean(strip[i], strip[j])
55                 count += 1
56                 if distance < distance_terdekat:
57                     distance_terdekat = distance
58                     pasangan_terdekat = (strip[i], strip[j])
59
60     return pasangan_terdekat, distance_terdekat, count
61
62 points_sorted = sorted(points, key=lambda p: p[0])
63 return dnc(points_sorted)
64

```

```

# Fungsi untuk membangkitkan titik secara acak
def acak_titik(n):
    points = []
    for i in range(n):
        x = random.randint(-1000, 1000)
        y = random.randint(-1000, 1000)
        z = random.randint(-1000, 1000)
        points.append((x, y, z))
    return points

# BONUS 1
# Fungsi untuk menggambar titik-titik dalam bidang 3D dan menandai sepasang titik terdekat
def plot_points(points, pasangan_terdekat=None):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    xs = [p[0] for p in points]
    ys = [p[1] for p in points]
    zs = [p[2] for p in points]
    ax.scatter(xs, ys, zs)
    if pasangan_terdekat:
        ax.scatter([pasangan_terdekat[0][0], pasangan_terdekat[1][0]], [pasangan_terdekat[0][1], pasangan_terdekat[1][1]], [pasangan_terdekat[0][2], pasangan_terdekat[1][2]])
        ax.plot([pasangan_terdekat[0][0], pasangan_terdekat[1][0]], [pasangan_terdekat[0][1], pasangan_terdekat[1][1]], [pasangan_terdekat[0][2], pasangan_terdekat[1][2]])
    plt.show()

# Main program
def dimensi_tiga():
    n = int(input("Masukkan banyaknya titik: "))
    points = acak_titik(n)

# Algoritma brute force
start_time = time.time()
brute_result, brute_distance, brute_count = brute_force(points)
brute_time = time.time() - start_time
print("\nHasil Brute Force:")
print("Sepasang titik terdekat:", brute_result)

```

```

# Main program
def dimensi_tiga():
    n = int(input("Masukkan banyaknya titik: "))
    points = acak_titik(n)

# Algoritma brute force
start_time = time.time()
brute_result, brute_distance, brute_count = brute_force(points)
brute_time = time.time() - start_time
print("\nHasil Brute Force:")
print("Sepasang titik terdekat:", brute_result)
print("Jarak terdekat:", brute_distance)
print("Banyaknya operasi perhitungan:", brute_count)
print("Waktu riil (detik):", brute_time)

# Algoritma divide and conquer
start_time = time.time()
dnc_result, dnc_distance, dnc_count = divide_and_conquer(points)
dnc_time = time.time() - start_time
print("\nHasil Divide and Conquer:")
print("Sepasang titik terdekat:", dnc_result)
print("Jarak terdekat:", dnc_distance)
print("Banyaknya operasi perhitungan:", dnc_count)
print("Waktu riil (detik):", dnc_time)
print("\n")

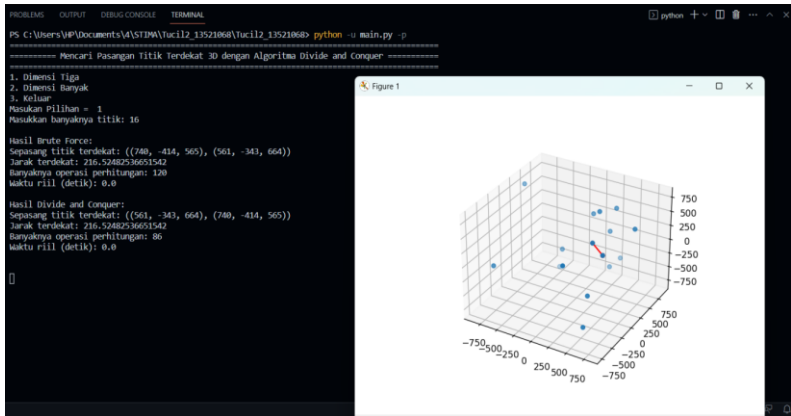
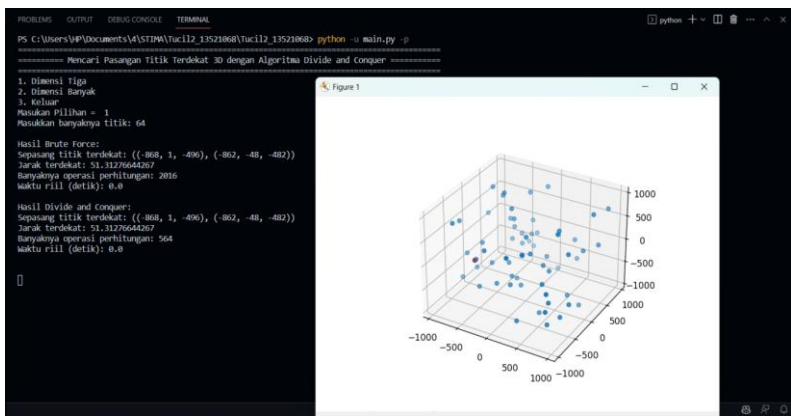
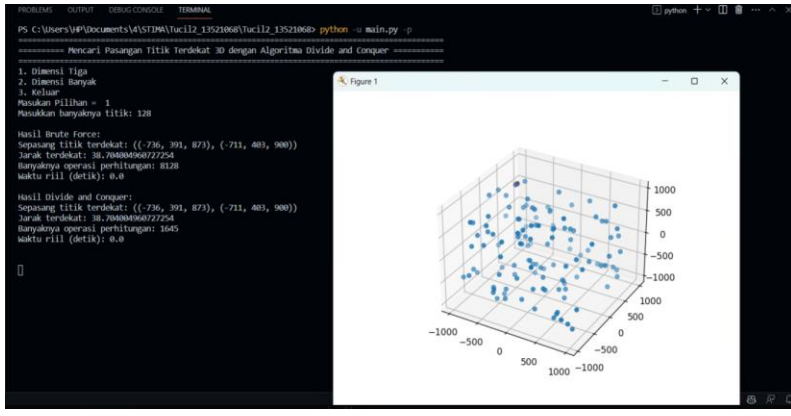
if n <= 1000:
    plot_points(points, pasangan_terdekat=dnc_result)
    plt.show()

```

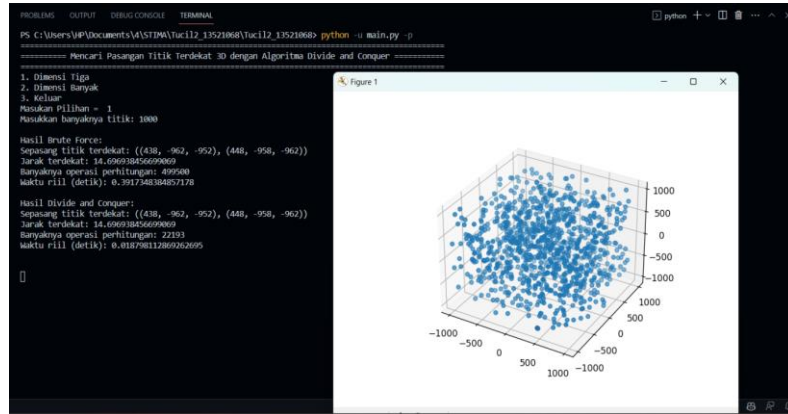
### 3. CONTOH MASUKAN DAN LUARAN

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	
64	
128	

1000



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

Test case (ukuran N)	Screenshot
16	<p>PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL</p> <pre>PS C:\Users\HP\Documents\ASTIMA\tucil2_13521068&gt; python -u main.py -p ===== Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer ===== 1. Dimensi Tiga 2. Dimensi Banyak 3. Keluar Masukkan Pilihan = 2 Masukkan jumlah titik : 16 Masukkan dimensi : 5  Hasil Brute Force: Sepasang titik terdekat: (array([ 624, 460, -127, 739, -100]), array([ 731, 160, 315, 899, -164])) Jarak terdekat: 571.4096604013621 Banyaknya operasi perhitungan: 120 Waktu riil (detik): 0.0  Hasil Divide and Conquer: Sepasang titik terdekat: (array([ 624, 460, -127, 739, -100]), array([ 731, 160, 315, 899, -164])) Jarak terdekat: 571.4096604013621 Banyaknya operasi perhitungan: 113 Waktu riil (detik): 0.0</pre>



64	<pre> PS C:\Users\IHP\Documents\4\STIMA\Tucil2_13521068\Tucil2_13521068&gt; python -u main.py -p ===== Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer =====  1. Dimensi Tiga 2. Dimensi Banyak 3. Keluar Masukan Pilihan = 2 Masukkan jumlah titik : 64 Masukkan dimensi : 5 Hasil Brute Force: Sepasang titik terdekat: (array([ 273, -345, 218, -808, 68]), array([ 162, -189, 239, -641, 51])) Jarak terdekat: 255.49168283918755 Banyaknya operasi perhitungan: 2016 Waktu riil (detik): 0.0  Hasil Divide and Conquer: Sepasang titik terdekat: (array([ 162, -189, 239, -641, 51]), array([ 273, -345, 218, -808, 68])) Jarak terdekat: 255.49168283918755 Banyaknya operasi perhitungan: 941 Waktu riil (detik): 0.017430782318115234  ===== Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer =====  1. Dimensi Tiga 2. Dimensi Banyak 3. Keluar Masukan Pilihan = 1 </pre>
128	<pre> PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1. Dimensi Tiga 2. Dimensi Banyak 3. Keluar Masukan Pilihan = 128 Pilihan yang benar!!!  ===== Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer =====  1. Dimensi Tiga 2. Dimensi Banyak 3. Keluar Masukan Pilihan = 2 Masukkan jumlah titik : 128 Masukkan dimensi : 5 Hasil Brute Force: Sepasang titik terdekat: (array([ 505, 680, 40, 423, -588]), array([ 488, 672, -78, 479, -714])) Jarak terdekat: 182.45273366517237 Banyaknya operasi perhitungan: 8128 Waktu riil (detik): 0.00687568099975586  Hasil Divide and Conquer: Sepasang titik terdekat: (array([ 488, 672, -78, 479, -714]), array([ 505, 680, 40, 423, -588])) Jarak terdekat: 182.45273366517237 Banyaknya operasi perhitungan: 2395 Waktu riil (detik): 0.018563508967426758  ===== Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer =====  1. Dimensi Tiga 2. Dimensi Banyak 3. Keluar Masukan Pilihan = 1 </pre>
1000	<pre> Sepasang titik terdekat: (array([ 734, 869, -272, 890, -341]), array([ 751, 805, -244, 783, -389])) Jarak terdekat: 137.55726080436466 Banyaknya operasi perhitungan: 33305 Waktu riil (detik): 0.08182501792907715  ===== Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer =====  1. Dimensi Tiga 2. Dimensi Banyak 3. Keluar Masukan Pilihan = 2 Masukkan jumlah titik : 1000 Masukkan dimensi : 5 Hasil Brute Force: Sepasang titik terdekat: (array([ 366, -921, 258, -5, 570]), array([ 386, -942, 362, 86, 576])) Jarak terdekat: 141.32948246105903 Banyaknya operasi perhitungan: 499500 Waktu riil (detik): 0.8788151741027832  Hasil Divide and Conquer: Sepasang titik terdekat: (array([ 366, -921, 258, -5, 570]), array([ 386, -942, 362, 86, 576])) Jarak terdekat: 141.32948246105903 Banyaknya operasi perhitungan: 33917 Waktu riil (detik): 0.08750009536743164 </pre>

#### 4. LINK GITHUB

Link Repo GitHub: [https://github.com/Ilhamgzzlr/Tucil2\\_13521068.git](https://github.com/Ilhamgzzlr/Tucil2_13521068.git)

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil <i>running</i>	✓	
3. Program dapat menerima masukan dan menuliskan keluaran	✓	
4. Keluaran program sudah benar (solusi <i>closest pair</i> benar)	✓	
5. Bonus 1 dikerjakan	✓	
6. Bonus 2 dikerjakan	✓	