LAPORAN TUGAS KECIL II STRATEGI ALGORITMA

MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA *DIVIDE AND CONQUER*

DISUSUN OLEH

NAMA: HAZIQ ABIYYU MAHDY

NIM: 13521170

KELAS: K02

PROGRAM STUDI TEKNIK INFORMATIKA

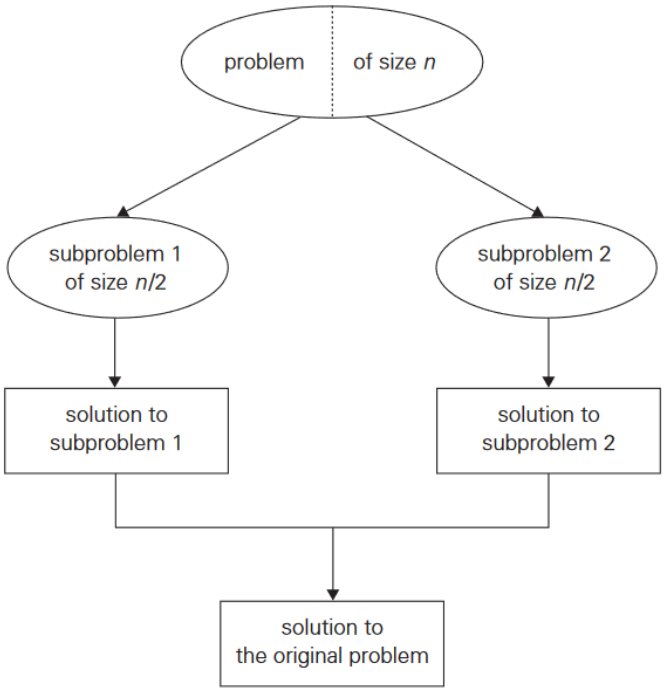
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BAB I
DESKRIPSI PERSOALAN

Algoritma *divide and conquer* adalah jenis algoritma yang menyelesaikan suatu permasalahan dengan membagi permasalahan tersebut menjadi upa-permasalahan yang lebih kecil. Secara umum, algoritma *divide and conquer* terdiri atas tiga langkah.

1. *Divide*, yaitu membagi permasalahan menjadi beberapa upa-permasalahan yang memiliki karakteristik yang sama dengan permasalahan semula. Idealnya, masing-masing upa-permasalahan memiliki ukuran yang sama.
2. *Conquer*, yaitu menyelesaikan upa-permasalahan, biasanya secara rekursif.
3. *Combine*, yaitu menggabungkan solusi dari upa-permasalahan untuk mendapatkan solusi permasalahan semula jika diperlukan.



Gambar 1.1 Ilustrasi algoritma *divide and conquer*

Pada tugas kecil ini, penulis mengembangkan algoritma *divide and conquer* dalam bahasa Python untuk mencari pasangan titik terdekat pada ruang 3D. Pendekatan *divide and conquer* tersebut kemudian dibandingkan dengan pendekatan *brute force* dari segi waktu eksekusi dan jumlah operasi yang dilakukan. Misalkan terdapat n buah titik pada ruang 3D. Setiap titik P di dalam ruang dinyatakan

dengan koordinat $P = (x, y, z)$. Jarak dua buah titk $P_1 = (x_1, y_1, z_1)$ dan $P_2 = (x_2, y_2, z_2)$ dihitung dengan rumus Euclidean berikut.

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

Pada tugas ini juga diberikan bonus berupa *plotting* titik-titik pada grafik (bonus 1) serta generalisasi program untuk dapat menemukan pasangan titik terdekat dalam R^N (bonus 2).

Masukan program:

1. Banyaknya titik (n)
2. Dimensi (untuk bonus 1)
3. Titik-titik (dibangkitkan secara acak)

Luaran program:

1. Pasangan titik yang jaraknya terdekat dan nilai jaraknya
2. Banyaknya operasi perhitungan rumus Euclidean
3. Waktu riil dalam detik serta spesifikasi komputer yang digunakan
4. Penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya (untuk bonus 2).

Pada program ini, digunakan beberapa *library* sebagai berikut.

1. math, untuk melakukan operasi sqrt()
2. numpy, untuk membangkitkan bilangan acak (pada fungsi random.uniform()) serta untuk struktur data NDArray
3. matplotlib, untuk melakukan *plotting* titik-titik pada grafik
4. time, untuk menghitung waktu eksekusi program.

BAB II

ALGORITMA PENYELESAIAN MASALAH

Diberikan array P[0..n-1] yang berisi titik-titik [p_0, p_1, .., p_{n-1}] dengan $p_i = (x_i, y_i,, z_i)$. Jika pencarian pasangan titik terdekat dilakukan dengan algoritma brute force, berikut adalah langkah-langkah yang dilakukan.

1. Simpan nilai jarak minimum sebesar tak hingga. Nilai ini kemudian akan diganti dengan jarak titik terdekat yang ada di P.
2. Lakukan iterasi terhadap seluruh titik-titik yang ada di P. Untuk setiap titik p_i, lakukan iterasi dari i+1 sampai n-1 dan lakukan perhitungan jarak Euclidean dengan seluruh titik [p_{i+1} .. p_{n-1}]. Jika terdapat pasangan titik yang jaraknya lebih kecil dibandingkan jarak minimum sebelumnya, maka pasangan titik tersebut di-*assign* sebagai pasangan terdekat dan jaraknya di-*assign* sebagai jarak minimum.
3. Setelah iterasi selesai, kembalikan pasangan titik terdekat dan jarak minimum.

Banyaknya operasi yang diperlukan untuk algoritma *brute force* adalah sebagai berikut.

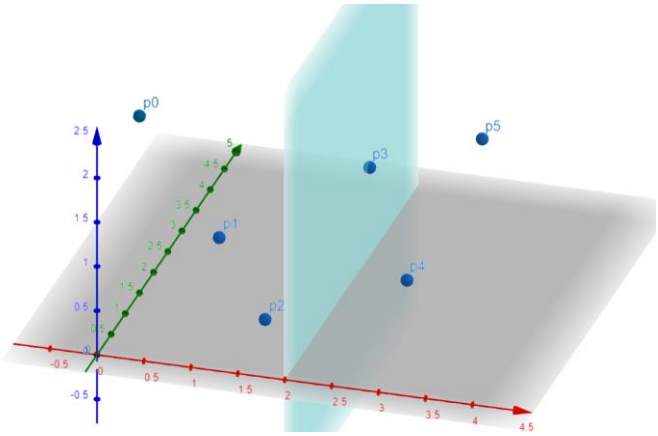
$$T(n) = \sum_{k=1}^{n} (n-k)$$

$$T(n) = \frac{n(n-1)}{2} = O(n^2)$$

Untuk menyelesaikan permasalahan yang sama dengan pendekatan *divide and conquer*, maka dilakukan langkah-langkah berikut.

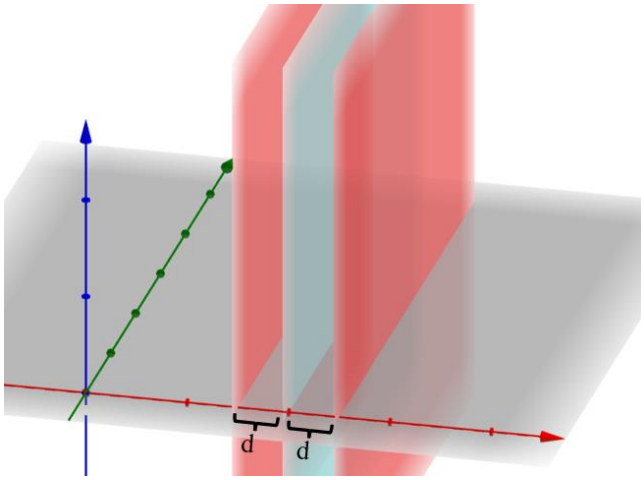
1. Titik-titik pada P diurutkan menaik berdasarkan nilai absis menggunakan algoritma *quicksort*
2. Kasus basis: apabila titik-titik yang ada pada P berjumlah dua atau tiga, maka lakukan pencarian pasangan titik terdekat dengan cara *brute force*.
3. Kasus rekurens: apabila titik-titik yang ada pada P berjumlah lebih dari tiga, ambil titik tengah yang dapat membagi *array* P menjadi dua bagian, yaitu partisi kiri dan pariti kanan. Titik tengah tersebut adalah $p_{n \text{ div } 2}$. Titik-titik [p_0 .. $p_{n \text{ div } 2 -1}$] berada di sebelah kiri $x_{n \text{ div } 2}$ dan titik-titik [$p_{n \text{ div } 2}$ .. p_n] berada di sebelah kanan $x_{n \text{ div } 2}$. Pada gambar dibawah, terdapat titik-titik [p_0 .. p_5], dengan

p_3 sebagai pembagi partisi. Bidang x = 2, yang merupakan absis dari titik p_3 membagi *array* menjadi dua partisi, dengan [p_0, p_1, p_2] sebagai partisi kiri dan [p_3, p_4, p_5] sebagai partisi kanan.



Gambar 2.1 Ilustrasi pembagian titik-titik menjadi dua partisi

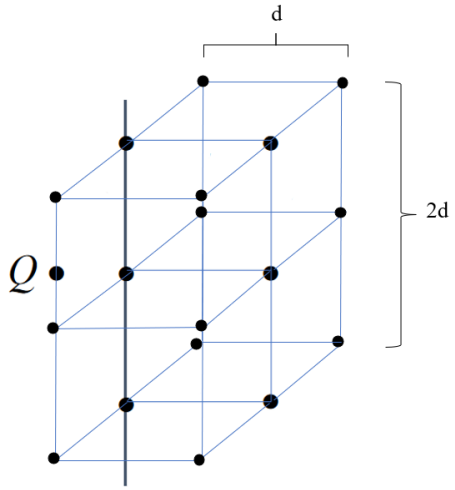
4. Lakukan pemanggilan fungsi *divide and conquer* untuk partisi kiri dan partisi kanan, kemudian cari jarak pasangan titik terdekat antara dua partisi tersebut, yaitu jarak terkecil di antara jarak terkecil pada partisi kiri dan jarak terkecil pada partisi kanan. Simpan jarak tersebut sebagai d dan simpan pula pasangan terdekat yang didapatkan.
5. Karena terdapat kemungkinan titik terdekat dipisahkan oleh bidang tengah, maka kita harus melakukan pengecekan terhadap titik-titik yang jarak absisnya dengan bidang tengah kurang dari atau sama dengan d. Daerah tersebut disebut *slab*.



Gambar 2.2 Ilustrasi *slab*

6. Kita dapat melakukan pengecekan terhadap jarak pasangan titik-titik yang ada pada *slab*. Apabila jarak suatu pasangan titik lebih kecil dari jarak minimum yang disimpan sebelumnya, maka kita akan memperbarui nilai jarak minimum dan pasangan terdekat. Namun, untuk sebuah titik Q (Q_x, Q_y, Q_z) di dalam *slab*, kita tidak perlu mencari jarak titik Q dengan seluruh titik lainnya pada *slab*. Terdapat paling banyak 18 titik yang perlu dicari jaraknya dengan titik Q, yaitu titik R (R_x, R_y, R_z) dengan $|Q_y - R_y| \leq d$ dan $|Q_z - R_z| \leq d$. Apabila $|Q_y - R_y| > d$ atau $|Q_z - R_z| > d$, maka jarak titik Q dan R > d, sehingga pasangan titik tersebut sudah pasti bukan

merupakan pasangan terdekat. Perlu diingat bahwa 18 hanyalah batas atas jumlah titik yang perlu dihitung jaraknya dengan Q. Pada umumnya, jumlah titik yang perlu ditinjau kurang dari 18. Misalkan Q sebuah titik pada bagian kiri *slab*, banyak titik maksimum yang perlu ditinjau dapat dilihat pada gambar berikut.



Gambar 2.3 Banyaknya titik maksimum yang perlu ditinjau dari Q

7. Setelah melakukan pencarian pada *slab*, kita akan mendapatkan pasangan titik terdekat serta jarak pasangan titik tersebut.

Pada kasus basis, banyaknya operasi perhitungan jarak Euclidean yang diperlukan pada pemanggilan fungsi *divide and conquer* adalah konstan (satu perhitungan untuk n = 2 atau tiga perhitungan untuk n = 3). Sedangkan pada kasus rekurens, banyaknya operasi yang diperlukan pada pemanggilan fungsi ini adalah dua kali banyaknya operasi yang diperlukan untuk memanggil fungsi yang sama untuk ukuran n/2 (pencarian pada partisi kiri dan partisi kanan) ditambah banyaknya operasi yang diperlukan untuk pencarian pada *slab*. Untuk setiap titik pada *slab*, batas atas banyaknya perhitungan jarak Euclidean yang diperlukan adalah 18 (suatu konstanta), maka dapat dinyatakan bahwa banyaknya operasi yang diperlukan untuk pencarian pada *slab* adalah kn, dengan k adalah suatu konstanta. Sehingga, banyaknya operasi yang diperlukan dapat dinyatakan sebagai fungsi berikut.

$$T(n) = 2T\left(\frac{n}{2}\right) + kn$$

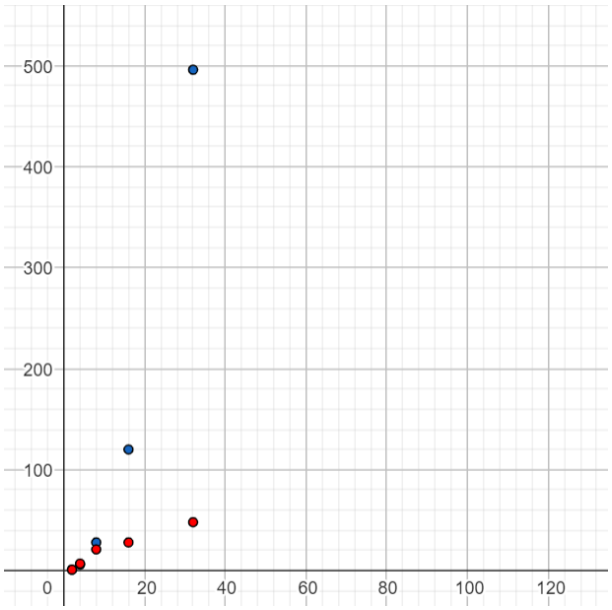
Kompleksitas dalam notasi *Big-Oh* dapat dihitung dengan menggunakan teorema Master.

$$T(n) \text{ adalah } \begin{cases} O(n^d) & \text{jika } a < b^d \\ O(n^d \log n) & \text{jika } a = b^d \\ O(n^{\log_b a}) & \text{jika } a > b^d \end{cases}$$

Diketahui a = 2, b = 2, dan d = 1. Karena a = b^d, maka kompleksitas algoritma ini adalah

$$O(n \log n)$$

Dapat dibuktikan bahwa pada permasalahan ini, algoritma *divide and conquer* secara teoretis lebih sangkil dibandingkan algoritma *brute force*. Untuk membuktikannya lebih lanjut, penulis melakukan *plotting* jumlah operasi perhitungan jarak Euclidean yang diperlukan pada kedua algoritma. Pada gambar di bawah, sumbu-x melambangkan banyaknya masukan dan sumbu-y melambangkan banyaknya operasi yang diperlukan. Dapat dilihat bahwa pertumbuhan jumlah operasi pada algoritma *divide and conquer* (dilambangkan dengan titik merah) jauh lebih lambat dibandingkan dengan pertumbuhan jumlah operasi pada algoritma *brute force* (dilambangkan dengan titik biru).



Gambar 2.4 Perbandingan algoritma *divide and conquer* dengan *brute force*

BAB III

SOURCE CODE

1. closest_pair.py

```
import math
import numpy as np
import matplotlib.pyplot as plt



def generate_random_points(N, dimensions, threshold=1000):
    """
    Mengembalikan numpy.NDArray berisi N buah titik acak dengan dimensi
sebesar dimensions.
    Seluruh titik akan berada pada range
    -1 * threshold <= x < threshold,
    -1 * threshold <= y < threshold,
    -1 * threshold <= z < threshold,
    dst.
    """
    arr_points = np.empty(N, dtype=object)

    for i in range(N):
        arr_points[i] = []

        for _ in range(dimensions):
            value = round(np.random.uniform(-1 * threshold, threshold), 2)
            arr_points[i].append(value)

        arr_points[i] = tuple(arr_points[i])

    return arr_points



def euclidean_dist(point_a, point_b, dimensions):
    """
    Mengembalikan jarak euclidean antara titik a dan titik b dengan dimensi
sebesar dimensions
    """
    result = 0
    for i in range(dimensions):
        result += (point_a[i] - point_b[i])**2

    result = math.sqrt(result)
    return result
```

```python
def bf_closest_pair(arr_points, dimensions):
    """
    Mengembalikan jarak terpendek, pasangan titik terdekat, serta banyaknya
operasi
    perhitungan jarak euclidean yang dilakukan dengan algoritma brute force
    Prekondisi:
    len(arr_points) >= 2 dan dimensions >=1
    """
    closest_pair = None
    min_dist = float('inf')
    operations = 0

    for i in range(0, len(arr_points)):
        for j in range(i+1, len(arr_points)):
            current_dist = euclidean_dist(arr_points[i], arr_points[j],
dimensions)
            operations += 1
            if (current_dist < min_dist):
                min_dist = current_dist
                closest_pair = (arr_points[i], arr_points[j])

    return min_dist, closest_pair, operations



def dnc_closest_pair(arr_points, dimensions):
    """
    Mengembalikan jarak terpendek, pasangan titik terdekat, serta banyaknya
operasi
    perhitungan jarak euclidean yang dilakukan dengan algoritma divide and
conquer.
    Prekondisi:
    len(arr_points) >= 2, dimensions >=1, dan arr_points
    sudah terurut berdasarkan absis
    """
    num_points = len(arr_points)

    if (num_points <= 3):
        return bf_closest_pair(arr_points, dimensions)

    # Pengambilan titik tengah yang dapat membagi array menjadi partisi kiri
dan kanan sama rata
    # dengan elemen tengah array sebagai acuan.
    mid = num_points // 2
    mid_x = arr_points[mid][0]
    left_arr_points = arr_points[:mid]
```

```
    right_arr_points = arr_points[mid:]

    # Pencarian pasangan terdekat pada partisi kiri dan kanan secara
rekursif
    left_min_dist, left_closest_pair, left_operations =
dnc_closest_pair(left_arr_points, dimensions)
    right_min_dist, right_closest_pair, right_operations =
dnc_closest_pair(right_arr_points, dimensions)

    # Jarak terpendek sementara adalah min(left_min_dist, right_min_dist)
    if (left_min_dist <= right_min_dist):
        closest_pair = left_closest_pair
        min_dist = left_min_dist
    else:
        closest_pair = right_closest_pair
        min_dist = right_min_dist

    # Titik-titik yang jarak absisnya dengan mid_x <= min_dist akan
dimasukkan ke dalam slab
    slab = []
    slab_width = min_dist
    slab_operations = 0

    for point in arr_points:
        if (abs(point[0] - mid_x) <= slab_width):
            slab.append(point)

    # Pencarian pasangan terdekat untuk titik-titik yang berada dalam slab
    for i in range(0, len(slab)):
        for j in range(i + 1, len(slab)):
            # Jika terdapat sepasang titik di dalam slab yang tidak memenuhi
kriteria close_enough
            # (lihat fungsi di bawah) maka tidak perlu dilakukan perhitungan
            # jarak euclidean, karena sudah pasti jaraknya > min_dist
            if (close_enough(slab[i], slab[j], slab_width, dimensions)):
                current_dist = euclidean_dist(slab[i], slab[j], dimensions)
                slab_operations += 1
                if (current_dist < min_dist):
                    min_dist = current_dist
                    closest_pair = (slab[i], slab[j])

    total_operations = left_operations + right_operations + slab_operations
    return min_dist, closest_pair, total_operations



def close_enough(point_a, point_b, dist_limit, dimensions):
    """
    Menentukan apakah titik a (x1, y1, z1) dan titik b (x2, y2, z2)
```

```
    cukup dekat dengan kriteria berikut:
    |x1 - x2| < dist_limit,
    |y1 - y2| < dist_limit,
    |z1 - z2| < dist_limit,
    dst.
    Digunakan untuk mengurangi jumlah perhitungan pada algoritma divide and
conquer
    """
    for i in range(1, dimensions):
        if (abs(point_a[i] - point_b[i]) > dist_limit):
            return False

    return True



def plot_results(arr_points, closest_pair):
    """
    Melakukan plotting terhadap titik-titik yang ada pada arr_points.
pasangan titik
    yang merupakan pasangan terdekat diberi warna merah, sedangkan yang
lainnya
    diberi warna hijau

    """
    fig = plt.figure(figsize = (16, 9))
    ax = plt.axes(projection ="3d")
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    for point in arr_points:
        if point in closest_pair:
            ax.scatter3D(point[0], point[1], point[2], color = "red")
        else:
            ax.scatter3D(point[0], point[1], point[2], color = "green")

    plt.show()



def quick_sort_by_x(arr_points, left, right):
    """
    Melakukan sorting pada arr_points dengan algoritma quick sort
    """
    if (left < right):
        pivot = arr_points[right][0]
        i = left - 1
        for j in range(left, right):
```

```
            if (arr_points[j][0] <= pivot):
                i += 1
                arr_points[i], arr_points[j] = arr_points[j], arr_points[i]

        pivotIdx = i + 1
        arr_points[pivotIdx], arr_points[right] = arr_points[right],
arr_points[pivotIdx]
        quick_sort_by_x(arr_points, left, pivotIdx-1)
        quick_sort_by_x(arr_points, pivotIdx + 1, right)
```

2. main.py

```
import time
import closest_pair as cp


print("====================== INPUTS ======================")
while (True):
    num_points = int(input("Number of points: "))
    if (num_points >=2):
        break
    print("Number of points must be greater than or equal to 2")

while (True):
    dim = int(input("Dimensions: "))
    if (dim >= 1):
        break
    print("Dimensions must be greater than or equal to 1")


print("====================================================")
print("\n")

arr_points = cp.generate_random_points(num_points, dim, threshold=1000)

bf_start_time = time.time()
bf_min_dist, bf_closest_pair, bf_operations = cp.bf_closest_pair(arr_points,
dim)
bf_finish_time = time.time()

print("==================== BRUTE FORCE ====================")
print("Min distance:", bf_min_dist)
print("Closest pair:", bf_closest_pair)
print("Execution time:", bf_finish_time - bf_start_time)
print("Total operations:", bf_operations)
print("====================================================")

print("\n")
```

```
dnc_start_time = time.time()
cp.quick_sort_by_x(arr_points, 0, len(arr_points) - 1)
dnc_min_dist, dnc_closest_pair, dnc_operations =
cp.dnc_closest_pair(arr_points, dim)
dnc_finish_time = time.time()

print("================ DIVIDE AND CONQUER =================")
print("Min distance:", dnc_min_dist)
print("Closest pair: ", dnc_closest_pair)
print("Execution time:", dnc_finish_time - dnc_start_time)
print("Total operations:", dnc_operations)
print("=====================================================")

if (dim == 3):
    ans = input("Plot results? (y/n): ")
    if (ans == "y"):
        cp.plot_results(arr_points, dnc_closest_pair)
```

BAB IV

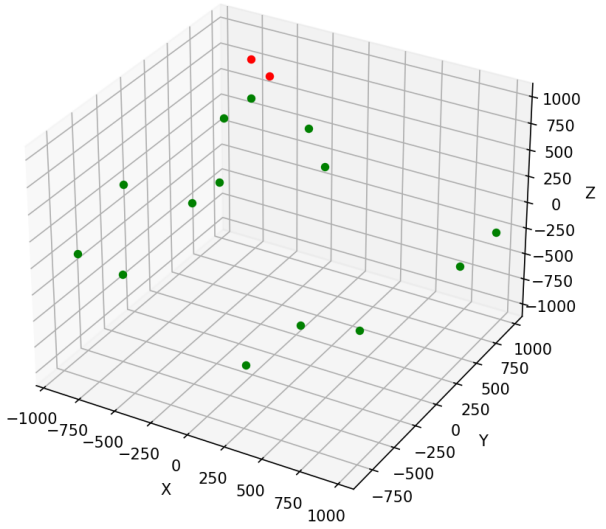
KASUS UJI DAN CHECKLIST

1. n = 16

```
====================== INPUTS ======================
Number of points: 16
Dimensions: 3
====================================================


=================== BRUTE FORCE ====================
Min distance: 173.9022731306293
Closest pair: ((-606.02, 734.19, 980.56), (-472.63, 722.97, 869.55))
Execution time: 0.0
Total operations: 120
====================================================


================ DIVIDE AND CONQUER ================
Min distance: 173.9022731306293
Closest pair:  ((-606.02, 734.19, 980.56), (-472.63, 722.97, 869.55))
Execution time: 0.0010089874267578125
Total operations: 20
====================================================
Plot results? (y/n): y
```

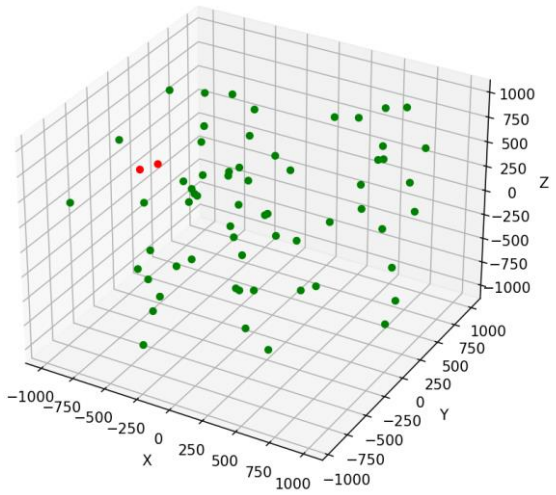


2. n = 64

```
====================== INPUTS ======================
Number of points: 64
Dimensions: 3
====================================================


=================== BRUTE FORCE ====================
Min distance: 123.43588902746237
Closest pair: ((-468.18, -429.27, 709.57), (-558.51, -503.4, 669.8))
Execution time: 0.0029973983764648438
Total operations: 2016
====================================================


================ DIVIDE AND CONQUER ================
Min distance: 123.43588902746237
Closest pair:  ((-558.51, -503.4, 669.8), (-468.18, -429.27, 709.57))
Execution time: 0.0010094642639160156
Total operations: 97
====================================================
Plot results? (y/n): y
```

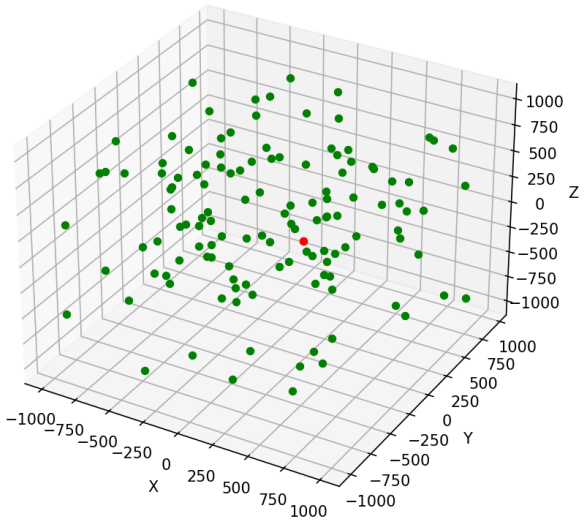


3. n = 128

```
====================== INPUTS ======================
Number of points: 128
Dimensions: 3
====================================================


=================== BRUTE FORCE ====================
Min distance: 45.83904122033966
Closest pair: ((-293.74, 917.74, -825.81), (-310.4, 947.63, -856.31))
Execution time: 0.008482217788696289
Total operations: 8128
====================================================


================ DIVIDE AND CONQUER ================
Min distance: 45.83904122033966
Closest pair:  ((-310.4, 947.63, -856.31), (-293.74, 917.74, -825.81))
Execution time: 0.0009984970092773438
Total operations: 198
====================================================
Plot results? (y/n): y
```

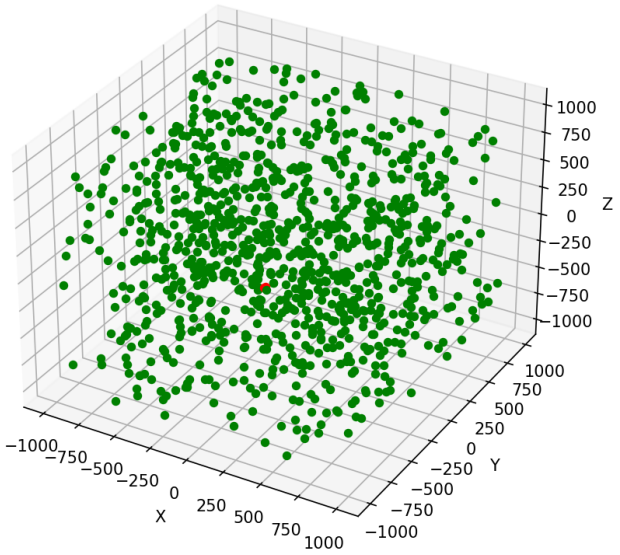


4. n = 1000

```
====================== INPUTS ======================
Number of points: 1000
Dimensions: 3
====================================================


=================== BRUTE FORCE ====================
Min distance: 8.966900244789194
Closest pair: ((128.67, -447.84, -0.38), (124.27, -455.61, -1.2))
Execution time: 0.5288677215576172
Total operations: 499500
====================================================


================ DIVIDE AND CONQUER ================
Min distance: 8.966900244789194
Closest pair:  ((124.27, -455.61, -1.2), (128.67, -447.84, -0.38))
Execution time: 0.024012327194213867
Total operations: 1682
====================================================
Plot results? (y/n): y
```



5. dimensi ≠ 3

```
====================== INPUTS ======================
Number of points: 256
Dimensions: 4
====================================================


==================== BRUTE FORCE ====================
Min distance: 64.90606597229568
Closest pair: ((357.07, 372.56, 663.55, -860.11), (335.71, 421.89, 653.13, -894.96))
Execution time: 0.03304457664489746
Total operations: 32640
====================================================


================ DIVIDE AND CONQUER =================
Min distance: 64.90606597229568
Closest pair:  ((335.71, 421.89, 653.13, -894.96), (357.07, 372.56, 663.55, -860.11))
Execution time: 0.005064487457275391
Total operations: 571
====================================================
```

```
==================== INPUTS ====================
Number of points: 773
Dimensions: 5
================================================


================== BRUTE FORCE ==================
Min distance: 107.3257499391455
Closest pair: ((716.25, -912.92, -395.59, -508.35, 221.54), (700.95, -881.38, -457.2, -576.65, 178.77))
Execution time: 0.28467655181884766
Total operations: 298378
================================================


=============== DIVIDE AND CONQUER ==============
Min distance: 107.3257499391455
Closest pair:  ((700.95, -881.38, -457.2, -576.65, 178.77), (716.25, -912.92, -395.59, -508.35, 221.54))
Execution time: 0.03922891616821289
Total operations: 1988
================================================
```

```
====================== INPUTS ======================
Number of points: 389
Dimensions: 1
====================================================


==================== BRUTE FORCE ====================
Min distance: 0.0
Closest pair: ((612.26,), (612.26,))
Execution time: 0.03808927536010742
Total operations: 75466
====================================================


================ DIVIDE AND CONQUER =================
Min distance: 0.0
Closest pair:  ((612.26,), (612.26,))
Execution time: 0.0010395050048828125
Total operations: 427
====================================================
```

BAB V

LINK TO REPOSITORY

https://github.com/haziqam/Tucil2_13521170

DAFTAR PUSTAKA

1. Levitin, A. (2014). *Introduction to the Design and Analysis of Algorithms: International Edition*. Pearson Education.
2. *3D scatterplot - Matplotlib 3.7.0 documentation*. Diakses dari https://matplotlib.org/stable/gallery/mplot3d/scatter3d.html pada 26 Februari 2023.
3. Munir, Rinaldi. 2021. Algoritma *Divide and Conquer* (Bagian 2). Bahan Kuliah IF2211 Strategi Algoritma. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy(2021)-Bag1.pdf pada 26 Februari 2023