IF2211 Strategi Algoritma

IMPLEMENTASI ALGORITMA DIVIDE AND CONQUER DALAM MENCARI PASANGAN TITIK TERDEKAT 3D

Laporan Tugas Kecil II

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma

Pada Semester 2 (dua) Tahun Akademik 2022/2023

Disusun Oleh:

Fakhri Muhammad Mahendra 13521045

Vanessa Rebecca Wiyono 13521151

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2023

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

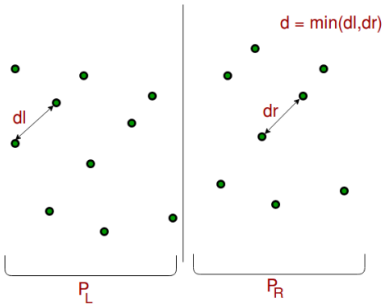
1.1 Algoritma *Divide and Conquer* pada *Closest Pair Problem*

Algoritma *Divide and Conquer* memecahkan suatu permasalahan dengan cara memecah atau membaginya menjadi beberapa bagian kecil sehingga akan lebih mudah untuk diselesaikan. Oleh karena itu, algoritma ini sering digunakan dalam memecahkan persoalan yang rumit. Berikut adalah langkah-langkah algoritma *Divide and Conquer* secara garis besar:

1. Divide : membagi masalah menjadi beberapa masalah yang lebih kecil
2. Conquer : Menyelesaikan setiap masalah kecil dan mendapatkan solusinya
3. Combine : menggabungkan solusi dari setiap masalah kecil untuk mendapat solusi secara keseluruhan

Closest Pair Problem merupakan masalah dalam komputasi geometri yang mencari pasangan titik terdekat diantara kumpulan titik dalam ruang n-dimensi. Pada umumnya, permasalahan ini digunakan dalam menyelesaikan masalah seperti navigasi, pencarian rute terpendek, dan pengenalan pola dalam data. Terdapat berbagai macam algoritma yang dapat digunakan untuk penyelesaiannya, dan salah satu alternatif algoritma yang dapat digunakan adalah algoritma *Divide and Conquer*. Berikut adalah langkah-langkahnya:

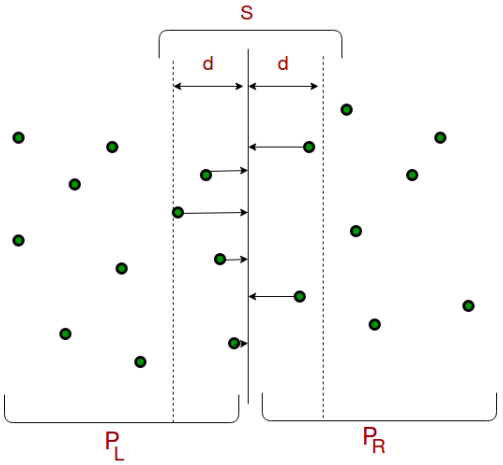
1. Urutkan titik-titik dalam himpunan terurut sesuai koordinat x sebagai *presort*
2. Sebagai base case, apabila titik yang di kurang dari sama dengan 3 maka gunakan algoritma brute force, yaitu mengecek semua kemungkinan pasangan titik, dan pilih yang jaraknya paling kecil.
3. Tentukan titik median dari himpunan yang sudah terurut berdasarkan sumbu x nya
4. Pecah himpunan titik menjadi dua bagian yang relatif sama besar dengan sumbu x *hyperplane* sama dengan sumbu x dari titik median
5. Selesaikan masalah secara rekursif pada dua himpunan titik yang sudah dibagi dua, akan didapat dua jarak terdekat dari dua himpunan titik tersebut



Sumber Gambar:

https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/

6. Semisal jarak terdekat antar pasangan titik dari himpunan pertama merupakan δ1 dan jarak terdekat dari himpunan kedua merupakan δ2, ambil nilai minimum dari kedua nya sehingga δ = **min**(δ1, δ2)
7. Kumpulkan titik-titik yang jaraknya paling jauh sebanyak δ dari *hyperplane* dan bagi titik-titik tersebut ke dua himpunan titik berdasarkan posisi relatif nya dari *hyperplane.* Apabila di kiri *hyperplane* masukan ke himpunan L, sebaliknya jika di kanan *hyperplane* masukan ke himpunan R



Sumber Gambar:

https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/

8. Akan dicari apakah terdapat pasangan titik dengan jarak dibawah δ dengan syarat satu titik berasal dari L, dan yang satu lagi berasal dari R. Hal tersebut dapat dilakukan dengan
    a. Untuk setiap titik di R, bandingkan dengan tiap titik yang ada di L. Semisal a ∈ R dan b ∈ R

b. Bandingkan apakah tiap nilai koordinat di titik a selain sumbu x, memiliki jarak yang lebih kecil sama dengan δ kepada tiap nilai koordinat di titik b selain sumbu x
    i. Semisal a = (1, 3, 2, 4), b = (2, 4, 2, -1), dan δ = 3. maka a dan b pada kasus ini tidak akan lolos uji tes jarak, karena nilai jarak antar sumbu koordinat ke-4 nya adalah |4 - (-1)| = 5 > 3 yaitu lebih dari δ
c. Apabila terdapat pasangan koordinat yang memenuhi uji tersebut, maka baru dihitung euclidean distance dari kedua titik tersebut dan dibandingkan dengan delta. Apabila lebih kecil, maka ganti nilai delta dengan nilai jarak baru yang lebih kecil

9. Ketika sudah didapat nilai jarak yang terkecil, kembalikan nilai tersebut untuk menyelesaikan rekursi

BAB II
SOURCE PROGRAM

2.1 array_of_points.py

```
import numpy as np
import random
from point import is_projection_close, euclidean_distance

def get_random_points(dimension, count):
    """Return array of random points, with each points have coordinate
    ranged between -100 and 100

    Args:
        dimension (int): dimension of points
        count (int): how many points to generate

    Returns:
        array of points:
    """
    points = []
    for i in range(count):
        point = []
        for j in range(dimension):
            coordinate = random.uniform(-100, 100)
            rounded_coordinate = round(coordinate, 2)
            point.append(rounded_coordinate)
        points.append(point)
```

```
    return points

def get_min_dist(min_dist, array_of_closest, point_1, point_2):
    """Compare and get minimum distance of current min_dist
    compared between distance between two points, update
    the accumulation array accordingly

    Args:
        min_dist (real): current minimum distance
        array_of_closest (array of pair of points): accumulation array
        point_1 (array of real):
        point_2 (array of real):

    Returns:
        tuple of minimum distance and accumulation array:
    """
    test_dist = euclidean_distance(point_1, point_2)
    if test_dist < min_dist:
        min_dist = test_dist
        array_of_closest = [[point_1, point_2]]
    elif test_dist == min_dist:
        array_of_closest.append([point_1, point_2])

    return min_dist, array_of_closest

def get_min_dist_from_2(min_dist_1, closest_point_1, min_dist_2, closest_point_2):
    """Get minumun distance from comparing two minimum distance candidate
    and two accumulation array

    Args:
        min_dist_1 (real):
        closest_point_1 (array of pair of points):
        min_dist_2 (real):
        closest_point_2 (array of pair of points):

    Returns:
        tuple of minimum distance and accumulation array:
    """
    if min_dist_1 < min_dist_2:
        min_dist = min_dist_1
        closest_points = closest_point_1
    elif min_dist_2 < min_dist_1:
        min_dist = min_dist_2
        closest_points = closest_point_2
    else:
        min_dist = min_dist_1
        closest_points = closest_point_1
        closest_points.extend(closest_point_2)

    return min_dist, closest_points

def sorted_arr_divider(sorted_arr):
```

```
    """I.S. input array is sorted
    divide an array to two relativly even number of element
    according to its x_axis value

    Args:
        sorted_arr (array of points): array already sorted increasing

    Returns:
        tuple of two divided array
    """
    size = np.shape(sorted_arr)[0]
    under_median = []
    over_median = []

    for i in range(size // 2):
        under_median.append(sorted_arr[i])

    for i in range(size // 2, size):
        over_median.append(sorted_arr[i])

    return (under_median, over_median)

def get_points_near_hyperplane(points, hp_axis, delta):
    """Get points that are atleast within distance of delta
    to hyperplane. Also group the array into two groups
    according to whether its left or right of hyperplane

    Args:
        points (array of points):
        hp_axis (real): hyperplane x axis value
        delta (real): minimum distance

    Returns:
        tuple of two divided array
    """
    size = len(points)
    left_hp = []
    right_hp = []

    for i in range(size):
        if hp_axis - delta <= points[i][0] < hp_axis:
            left_hp.append(points[i])
        elif hp_axis <= points[i][0] <= hp_axis + delta:
            right_hp.append(points[i])

    return left_hp, right_hp

def get_closest_near_hyperplane(left_arr, right_arr, min_dist, array_of_closest):
    size_left = len(left_arr)
    size_right = len(right_arr)
    for i in range(size_left):
        for j in range(size_right):
```

```
            if is_projection_close(left_arr[i], right_arr[j], min_dist):
                min_dist, array_of_closest = get_min_dist(min_dist, array_of_closest,
left_arr[i], right_arr[j])

    return min_dist, array_of_closest
```

2.2 brute_force.py

```
import math
import random
from visualization import*
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sympy import false, true

def jarak(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2 + (p1[2]
- p2[2])**2)

def tuple_pair_points(arrpoint1, arrpoint2):
        pairs = [(arrpoint1[i], arrpoint2[i]) for i in range
(len(arrpoint1))]
    return pairs

def closest_points(points):
    n = len(points)
    min_dist = float('inf')
    closest_p1, closest_p2 = None, None

    arrPoints1 = []
    arrPoints2 = []
    for i in range(n):
        for j in range(i+1, n):
            dist = jarak(points[i], points[j])
            if dist < min_dist:
                min_dist = dist
                closest_p1, closest_p2 = points[i], points[j]
                arrPoints1.append(closest_p1)
                arrPoints2.append(closest_p2)
```

```
            elif dist == min_dist:
                closest_p1, closest_p2 = points[i], points[j]
                arrPoints1.append(closest_p1)
                arrPoints2.append(closest_p2)

    return arrPoints1, arrPoints2, closest_p1, closest_p2, min_dist

def random_point():
    points = []
    n = random.randint(2, 100)
    for i in range(n):
        x = random.uniform(0, 1)
        y = random.uniform(0, 1)
        z = random.uniform(0, 1)
        point = (x, y, z)
        points.append(point)
    return points


#points = random_point()
#arrPoints1,   arrPoints2,   closest_p1,   closest_p2,   min_dist   =
closest_points(points)
#arrPoints = arrPoints1 + arrPoints2

#for i in range(len(arrPoints1)):
       #print("Titik  terdekat  ",  i+1,  ":  ",  arrPoints1[i],  "dan",
arrPoints2[i])
    #print("dengan jarak", min_dist)

#print(min_dist)
#visualization(points, arrPoints)
```

2.3 divide_conquer.py

```
      import numpy as np
      from       array_of_points       import       sorted_arr_divider,
   get_min_dist_from_2,                    get_points_near_hyperplane,
   get_closest_near_hyperplane
      from brute_force import brute_force_closest_pair

      def dnc_closest_pair(arr_points):
```

```
        size = len(arr_points)

        # Base case, if only 3 points or less, just use brute force approach
        if size <= 3:
            return brute_force_closest_pair(arr_points)

        # Divide array of points, according to x axis, less than median go to left array
        # more than median go to right array
        left_points, right_points = sorted_arr_divider(arr_points)

        # Recursively vall divide and conquer algorithm to solver for each sides
                    left_min_dist,    left_closest_points        = dnc_closest_pair(left_points)
                            right_min_dist,     right_closest_points= dnc_closest_pair(right_points)

        # Compare result for each sides and get the minimum distance and the corresponding pairs of point
                                min_dist,       closest_points      = get_min_dist_from_2(left_min_dist,            left_closest_points, right_min_dist, right_closest_points)

        # Get median to determine x axis of hyperplane
        x_median = arr_points[size//2][0]

        # Get points that at most minimum_distance near hyperplane
        left_hp, right_hp = get_points_near_hyperplane(arr_points, x_median, min_dist)

        # Compare current min_dist to the minimum distance from points near hyperplane
                                min_dist,       array_of_closest      = get_closest_near_hyperplane(left_hp,      right_hp,      min_dist, closest_points)

        return min_dist, array_of_closest
```

2.4 interface.py

```
    def get_int_input():

        while True:
            num = input("Masukan: ")
            try:
                val = int(num)
                return val
            except ValueError:
                try:
                    float(num)
                    print("\nMasukan tidak boleh bilangan desimal")
                    print("Silahkan ulangi kembali")
                except ValueError:
                    print("\nMasukan harus berupa bilangan bulat")
                    print("Silahkan ulangi kembali")

    def get_dimension_and_n():
        while True:
            print("Masukan dimensi titik")
            dimension = get_int_input()
            if dimension >= 3:
                break
             print("\nDimensi harus bernilai lebih dari sama dengan
3")

        print("")

        while True:
            print("Masukan jumlah titik")
            points_count = get_int_input()
            if points_count >= 2:
                break
             print("\nDimensi harus bernilai lebih dari sama dengan
2")

        return dimension, points_count

    def    output_format(time,    min_distance,    euclidean_count,
solution_array):
        print(f"Waktu dibutuhkan                    : {time}")
           print(f"Jarak  titik  terdekat                          :
{min_distance}")
```

```
                print(f"Operasi euclidean distance sebanyak :
{euclidean_count}")
        print("Pasangan titik:")
        for i in range(len(solution_array)):
            print(solution_array[i])
```

2.5 main.py

```
    from interface import get_dimension_and_n, output_format
    from array_of_points import get_random_points
    from brute_force import brute_force_closest_pair
    from quicksort import quicksort
    from divide_conquer import dnc_closest_pair
    from visualization import visualization
    import time

    def main():
        # Get input from user
        print("Selamat datang di closest pair finder")
        dimension, n = get_dimension_and_n()

        # Generate random points
        arr_points = get_random_points(dimension, n)
        arr_points = [[1,1,1], [2,2,2],[4,4,4], [5,5,5]]

        # Find closest pair with brute force
        time_start = time.time()
                        bf_min_distance,     bf_solution_pairs     =
brute_force_closest_pair(arr_points)
        time_finish = time.time()

        bf_time = time_finish - time_start
        from point import euclidean_count as bf_euclidean_count

        # Output answer brute force
        print("\nDengan pendekatan brute force:")
         output_format(bf_time, bf_min_distance, bf_euclidean_count,
bf_solution_pairs)


        # Find closest pair with divide and conquer
```

```
        time_start = time.time()
        arr_points = quicksort(arr_points)
                        dnc_min_distance,     dnc_solution_pairs     =
dnc_closest_pair(arr_points)
        time_finish = time.time()

        dnc_time = time_finish - time_start
        from point import euclidean_count
        dnc_euclidean_count = euclidean_count - bf_euclidean_count

        # Output answer divide and conquer
        print("\nDengan pendekatan divide and conquer")
                                                output_format(dnc_time,
dnc_min_distance,dnc_euclidean_count, dnc_solution_pairs)

        # Get visualization if points in 3D
        if dimension == 3:
            visualization(arr_points, bf_solution_pairs)


    if __name__ == '__main__':
        main()
```

2.6 point.py

```
    import numpy as np

    euclidean_count = 0

    def euclidean_distance(point_1, point_2):
        global euclidean_count
        euclidean_count += 1
        point_1 = np.asarray(point_1)
        point_2 = np.asarray(point_2)
        difference_res = np.subtract(point_1, point_2)
        squared_res = np.power(difference_res, 2)
        sum_res = np.sum(squared_res)
        return np.sqrt(sum_res)

    def is_projection_close(point_1, point_2, delta) -> bool:
        dimension = len(point_1)
```

```
        for i in range(1, dimension):
            if (abs(point_1[i] - point_2[i]) > delta):
                return False

        return True




    # print(euclidean_distance([1,1], [2,2]))
```

2.7 quicksort.py

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        left = []
        right = []
        for i in range(1, len(arr)):
            if arr[i][0] < pivot[0]:
                left.append(arr[i])
            else:
                right.append(arr[i])
        return quicksort(left) + [pivot] + quicksort(right)
```

2.8 visualization.py

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def visualization(points, solution_pairs):

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    for point in points:
        ax.scatter(point[0], point[1], point[2], c='black')

    for i, point in enumerate(solution_pairs):
        if (i % 3) == 0:
            ax.scatter(point[0][0], point[0][1], point[0][2], c='blue')
            ax.scatter(point[1][0], point[1][1], point[1][2], c='blue')
```

```
        elif (i % 3) == 1:
            ax.scatter(point[0][0], point[0][1], point[0][2], c='green')
            ax.scatter(point[1][0], point[1][1], point[1][2], c='green')
        else:
            ax.scatter(point[0][0], point[0][1], point[0][2], c='red')
            ax.scatter(point[1][0], point[1][1], point[1][2], c='red')

        x = [point[0][0], point[1][0]]
        y = [point[0][1], point[1][1]]
        z = [point[0][2], point[1][2]]

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

    ax.set_title("Scatter Plot of {} Random Points in 3D".format(len(points)))
    ax.set_xlabel("X Coordinates")
    ax.set_ylabel("Y Coordinates")
    ax.set_zlabel("Z Coordinates")

    # Display the plot
    plt.show()
```

BAB III

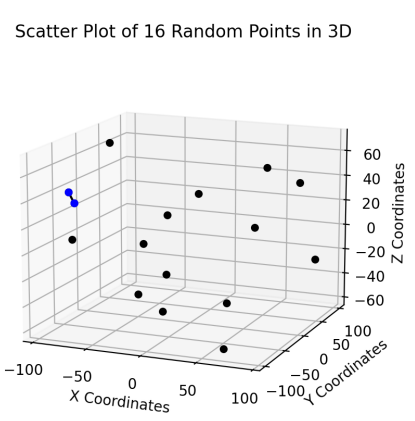
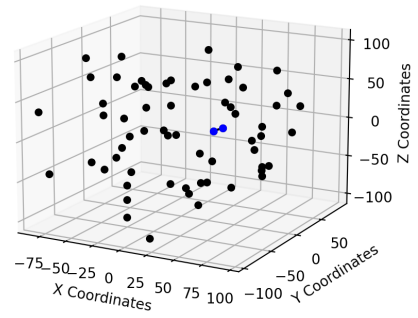
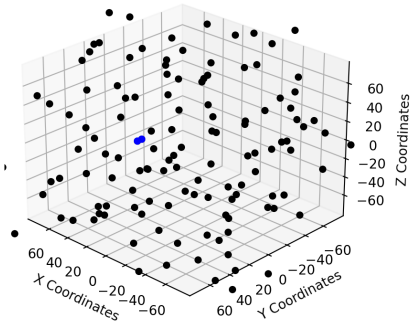
TEST CASE

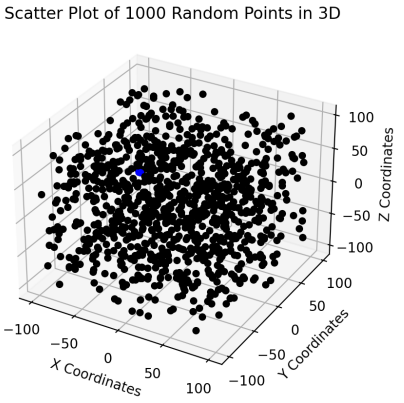
Test case dijalankan pada sistem operasi Windows, dengan laptop yang memiliki spesifikasi:

- Processor : 11th Gen Intel(R) Core(TM) i7-11370H @ 3.30GHz 3.30 GHz
- RAM : 16,0 GB (15,8 GB usable)
- Windows : Windows 10 Home Single Language

3.1 Test Case Ruang 3 Dimensi

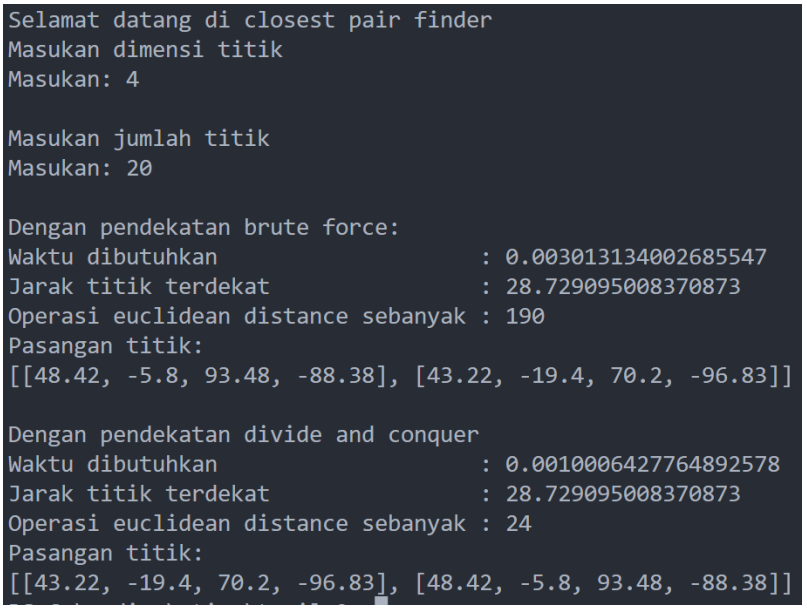
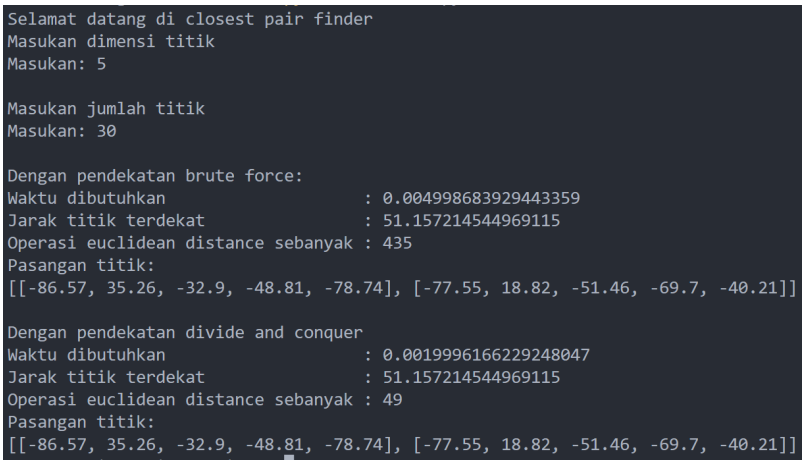
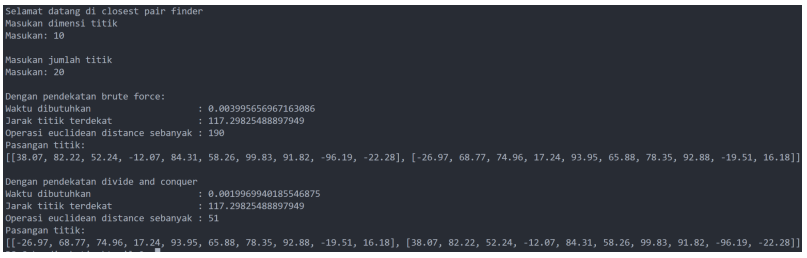
N	Output Hasil Uji	Visualisasi

16	Selamat datang di closest pair finder Masukan dimensi titik Masukan: 3 Masukan jumlah titik Masukan: 16 Dengan pendekatan brute force: Waktu dibutuhkan : 0.0019872188568115234 Jarak titik terdekat : 11.066892065977692 Operasi euclidean distance sebanyak : 120 Pasangan titik: [[-81.2, -77.3, 45.16], [-78.19, -72.22, 35.8]] Dengan pendekatan divide and conquer Waktu dibutuhkan : 0.00099945068359375 Jarak titik terdekat : 11.066892065977692 Operasi euclidean distance sebanyak : 15 Pasangan titik: [[-81.2, -77.3, 45.16], [-78.19, -72.22, 35.8]]	
64	Selamat datang di closest pair finder Masukan dimensi titik Masukan: 3 Masukan jumlah titik Masukan: 64 Dengan pendekatan brute force: Waktu dibutuhkan : 0.03211522102355957 Jarak titik terdekat : 8.295450560397553 Operasi euclidean distance sebanyak : 2016 Pasangan titik: [[79.39, -81.26, 42.85], [72.8, -84.68, 39.15]] Dengan pendekatan divide and conquer Waktu dibutuhkan : 0.0020127296447753906 Jarak titik terdekat : 8.295450560397553 Operasi euclidean distance sebanyak : 51 Pasangan titik: [[72.8, -84.68, 39.15], [79.39, -81.26, 42.85]]	
128	Selamat datang di closest pair finder Masukan dimensi titik Masukan: 3 Masukan jumlah titik Masukan: 128 Dengan pendekatan brute force: Waktu dibutuhkan : 0.07968854904174805 Jarak titik terdekat : 3.5130755756174734 Operasi euclidean distance sebanyak : 8128 Pasangan titik: [[-1.68, 45.05, 24.57], [-0.43, 48.09, 23.33]] Dengan pendekatan divide and conquer Waktu dibutuhkan : 0.0029876232147216797 Jarak titik terdekat : 3.5130755756174734 Operasi euclidean distance sebanyak : 140 Pasangan titik: [[-1.68, 45.05, 24.57], [-0.43, 48.09, 23.33]]	

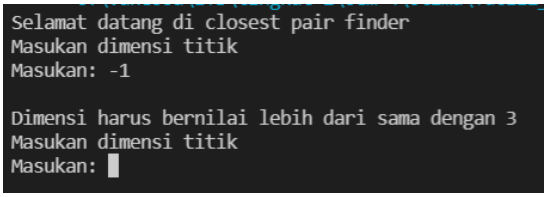
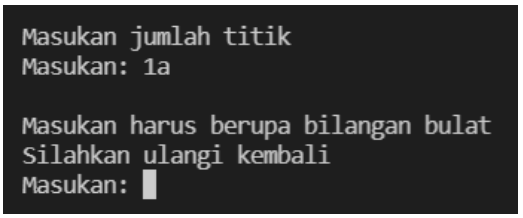
1000	Selamat datang di closest pair finder Masukan dimensi titik Masukan: 3 Masukan jumlah titik Masukan: 1000 Dengan pendekatan brute force: Waktu dibutuhkan : 4.98111629486084 Jarak titik terdekat : 1.8059346610550462 Operasi euclidean distance sebanyak : 499500 Pasangan titik: [[-27.78, -27.06, 52.09], [-26.43, -26.01, 51.51]] Dengan pendekatan divide and conquer Waktu dibutuhkan : 0.042208194732666016 Jarak titik terdekat : 1.8059346610550462 Operasi euclidean distance sebanyak : 1002 Pasangan titik: [[-27.78, -27.06, 52.09], [-26.43, -26.01, 51.51]]	

3.2. Test Case Ruang N Dimensi

N	Dimensi	Output Hasil Uji
100	1	Selamat datang di closest pair finder Masukan dimensi titik Masukan: 1 Masukan jumlah titik Masukan: 100 Dengan pendekatan brute force: Waktu dibutuhkan : 0.05465245246887207 Jarak titik terdekat : 0.010000000000005116 Operasi euclidean distance sebanyak : 4950 Pasangan titik: [[62.66], [62.67]] Dengan pendekatan divide and conquer Waktu dibutuhkan : 0.0019872188568115234 Jarak titik terdekat : 0.010000000000005116 Operasi euclidean distance sebanyak : 98 Pasangan titik: [[62.66], [62.67]]

20	4	 Selamat datang di closest pair finder Masukan dimensi titik Masukan: 4 Masukan jumlah titik Masukan: 20 Dengan pendekatan brute force: Waktu dibutuhkan : 0.003013134002685547 Jarak titik terdekat : 28.729095008370873 Operasi euclidean distance sebanyak : 190 Pasangan titik: [[48.42, -5.8, 93.48, -88.38], [43.22, -19.4, 70.2, -96.83]] Dengan pendekatan divide and conquer Waktu dibutuhkan : 0.0010006427764892578 Jarak titik terdekat : 28.729095008370873 Operasi euclidean distance sebanyak : 24 Pasangan titik: [[43.22, -19.4, 70.2, -96.83], [48.42, -5.8, 93.48, -88.38]]
5	10	 Selamat datang di closest pair finder Masukan dimensi titik Masukan: 5 Masukan jumlah titik Masukan: 30 Dengan pendekatan brute force: Waktu dibutuhkan : 0.004998683929443359 Jarak titik terdekat : 51.157214544969115 Operasi euclidean distance sebanyak : 435 Pasangan titik: [[-86.57, 35.26, -32.9, -48.81, -78.74], [-77.55, 18.82, -51.46, -69.7, -40.21]] Dengan pendekatan divide and conquer Waktu dibutuhkan : 0.0019996166229248047 Jarak titik terdekat : 51.157214544969115 Operasi euclidean distance sebanyak : 49 Pasangan titik: [[-86.57, 35.26, -32.9, -48.81, -78.74], [-77.55, 18.82, -51.46, -69.7, -40.21]]
20	10	 Selamat datang di closest pair finder Masukan dimensi titik Masukan: 10 Masukan jumlah titik Masukan: 20 Dengan pendekatan brute force: Waktu dibutuhkan : 0.003995656967163086 Jarak titik terdekat : 117.29825488897949 Operasi euclidean distance sebanyak : 190 Pasangan titik: [[38.07, 82.22, 52.24, -12.07, 84.31, 58.26, 99.83, 91.82, -96.19, -22.28], [-26.97, 68.77, 74.96, 17.24, 93.95, 65.88, 78.35, 92.88, -19.51, 16.18]] Dengan pendekatan divide and conquer Waktu dibutuhkan : 0.0019969940185546875 Jarak titik terdekat : 117.29825488897949 Operasi euclidean distance sebanyak : 51 Pasangan titik: [[-26.97, 68.77, 74.96, 17.24, 93.95, 65.88, 78.35, 92.88, -19.51, 16.18], [38.07, 82.22, 52.24, -12.07, 84.31, 58.26, 99.83, 91.82, -96.19, -22.28]]

3.3. Test Case Input Tidak Valid

N	Dimensi	Output Hasil Uji
	-1	
1a		

BAB IV
DAFTAR PUSTAKA

GeeksforGeeks, url: https://www.geeksforgeeks.org/divide-and-conquer/

Munir, Rinaldi. Algoritma Divide and Conquer (Bagian 1). 2023. url: https :/ / informatika . stei.itb.ac . id / ~rinaldi . munir / Stmik / 2020 - 2021 / Algoritma-Divide -and-Conquer-(2021)-Bagian1.pdf

Munir, Rinaldi. Algoritma Divide and Conquer (Bagian 1). 2023. url: https :/ / informatika . stei.itb.ac . id / ~rinaldi . munir / Stmik / 2020 - 2021 / Algoritma-Divide -and-Conquer-(2021)-Bagian2.pdf

Munir, Rinaldi. Algoritma Divide and Conquer (Bagian 1). 2023. url: https :/ / informatika . stei.itb.ac . id / ~rinaldi . munir / Stmik / 2020 - 2021 / Algoritma-Divide -and-Conquer-(2021)-Bagian3.pdf

Munir, Rinaldi. Algoritma Divide and Conquer (Bagian 1). 2023. url: https :/ / informatika . stei.itb.ac . id / ~rinaldi . munir / Stmik / 2020 - 2021 / Algoritma-Divide -and-Conquer-(2021)-Bagian4.pdf

USCSB, url: https://sites.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf

LAMPIRAN

Repository GitHub:
https://github.com/vanessrw/Tucil2_13521045_13521151

Checklist Table

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