Laporan Tugas Kecil II
IF2211 Strategi Algoritma
Mencari Pasangan Titik Terdekat 3D dengan Algoritma *Divide and Conquer*

Disusun Oleh :

Yanuar Sano Nur Rasyid	13521110
Febryan Arota Hia	13521120

1. Deskripsi Masalah

Mencari pasangan titik terdekat dengan Algoritma Divide and Conquer sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Pada Tucil 2 kali ini Anda diminta mengembangkan algoritma mencari pasangan titik terdekat pada bidang 3D. Misalkan terdapat n buah titik pada ruang 3D. Setiap titik P di dalam ruang dinyatakan dengan koordinat P = (x, y, z). Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titk P1 = (x1, y1, z1) dan P2 = (x2, y2, z2) dihitung dengan rumus Euclidean berikut:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Masukan program:

- n
- titik-titik (dibangkitkan secara acak) dalam koordinat (x, y, z)

Luaran program

- sepasang titik yang jaraknya terdekat dan nilai jaraknya
- banyaknya operasi perhitungan rumus Euclidean
- waktu riil dalam detik (spesifikasikan komputer yang digunakan)

2. Algoritma

Langkah pertama, periksa berapakah jumlah n atau banyak titik yang ada. Apabila $n \leq 3$, gunakan algoritma *brute force* untuk menyelesaikannya. Untuk n = 2 akan dijadikan sebagai basis genap sedangkan n = 3 dijadikan sebagai basis ganjil karena 3 tidak dapat lagi dibagi dua.

Berikutnya, apabila $n > 3$, program akan mengurutkan titik-titik berdasarkan koordinat x menggunakan fungsi *quick sort* atau *merge sort* (Pemilihan metode sorting dapat diatur pada file main.py). Titik-titik yang telah diurutkan tersebut kemudian dibagi menjadi dua bagian sama banyak (*left* dan *right*). Apabila banyak titik ganjil, maka letakkan lebih banyak satu titik di salah satu bagian.

Selanjutnya, gunakan rekursif pada masing-masing bagian yang telah dibagi tadi untuk mendapatkan pasangan titik terdekat pada kedua bagian. Ambil jarak terkecil dari kedua solusi rekursif tersebut, simpan pada sebuah variable resDist (*result distance*). Setelah itu cari titik-titik yang memiliki jarak kurang dari atau sama dengan resDist dari garis pembagi antara kedua bagian. Hal ini dilakukan untuk memeriksa kemungkinan terdapat pasangan terdekat pada *strip area*. Kemudian urutkan titik-titik tersebut berdasarkan koordinat y.

Untuk setiap titik yang telah diurutkan, periksa pasangan titik yang mempunyai jarak kurang dari dari resDist dengan syarat range sumbu x atau sumbu z lebih besar dari resDist. Ubah nilai redDist apabila ada pasangan titik yang lebih dekat. Terakhir, kembalikan pasangan titik terdekat beserta jaraknya. Jumlah titik yang diperiksa dapat disederhanakan menjadi O(1) dengan memeriksa titik-titik yang mungkin dapat bernilai lebih kecil saja. Penentuan jumlah dari titik tersebut dapat dihitung dari $c.4^d$ dengan c adalah invers dari konstanta volume bola pada d dan d adalah dimensi dari poin.

3. Source Code

File sort.py

```
from point import *

def merge_sort(points: list, n: int) -> list:
    if len(points) <= 1:
        return points

    mid = len(points) // 2
    left = merge_sort(points[:mid], n)
    right = merge_sort(points[mid:], n)

    return merge(left, right, n)


def merge(left: list, right: list, n: int) -> list:
    result = []
    i = 0
    j = 0
    while i < len(left) and j < len(right):
        if left[i].c[n] <= right[j].c[n]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result += left[i:]
    result += right[j:]

    return result


def quicksort(points: list, n : int) -> list:
    if len(points) <= 1:
        return points

    pivot = points[len(points) // 2]
```

```
    left = [p for p in points if p.c[n] < pivot.c[n]]
    middle = [p for p in points if p.c[n] == pivot.c[n]]
    right = [p for p in points if p.c[n] > pivot.c[n]]

    return quicksort(left, n) + middle + quicksort(right, n)
```

Function/Method	Keterangan
merge_sort(points: list, n: int) -> list	Mengurutkan list of points berdasarkan sumbu ke-n menggunakan metode *merge sort*
quicksort(points: list, n : int) -> list	Mengurutkan list of points berdasarkan sumbu ke-n menggunakan metode *quick sort*

File point.py

```
import math
import random

class Point:
    def __init__(self, coords: list):
        self.c = coords
        self.d = len(coords)

    def print_point(self):
        print("(", end="")
        for i in range(self.d):
            if i == self.d - 1:
                print(f"{self.c[i]})")
            else:
                print(f"{self.c[i]}; " , end="")

def random_point(number_of_point: int, dimension: int, rounding:
int = 3, limit: int = 100) -> list:
  points = []
  for _ in range(number_of_point):
    point = []
    for _ in range(dimension):
      point.append(round(random.uniform(-limit, limit), rounding))
```

```
    points.append(Point(point))
  return points

def print_points(Points: list):
  for i in range (len(Points)):
    Points[i].print_point()
```

Function/Method	Keterangan
__init__(self, coords: list)	Constructor kelas point
print_point(self)	Mencetak point ke layar
random_point(number_of_point: int, dimension: int, rounding: int = 3, limit: int = 100) -> list	Men-*generate* list of point dengan dimensi tertentu dan limit yang ditentukan
print_points(Points: list)	Mencetak seluruh point pada list ke layar

Variable/Attribute	Keterangan
self.c = coords	Sebuah list dengan elemen tiap indeks menyatakan nilai koordinat
self.d = len(coords)	Angka yang menyatakan banyak nilai koordinat atau dimensi

File plotPoint.py

```
import matplotlib.pyplot as plt

def plotPoint(points: list, result: list, d: int):

    fig = plt.figure()
    if d == 1:
        for point in points:
            if (point == result[0] or point == result[1]):
                plt.scatter(point.c[0], 0, c = 'r')
            else :
```

```
                plt.scatter(point.c[0], 0, c = 'b')

    elif d == 2:
        for point in points:
            if (point == result[0] or point == result[1]):
                plt.scatter(point.c[0], point.c[1], c = 'r')
                plt.text(point.c[0], point.c[1], f"{point.c[0]},
{point.c[1]}", color='k')
            else :
                plt.scatter(point.c[0], point.c[1], c = 'b')

    elif d == 3:
        ax = fig.add_subplot(111, projection='3d')
        for point in points:
            if (point == result[0] or point == result[1]):
                ax.scatter(point.c[0], point.c[1], point.c[2], c =
'r')
                ax.text(point.c[0], point.c[1], point.c[2],
f"{point.c[0]}, {point.c[1]}, {point.c[2]}", color='k')
            else :
                ax.scatter(point.c[0], point.c[1], point.c[2], c =
'b')

    elif d == 4:
        x = []
        y = []
        z = []
        c = []
        for point in points:
            x.append(point.c[0])
            y.append(point.c[1])
            z.append(point.c[2])
            c.append(point.c[3])

        ax = fig.add_subplot(111, projection='3d')
        img = ax.scatter(x, y, z, c=c, cmap=plt.hot())
        for i in range(len(points)):
            for j in range (len(result)):
                if result[j].c[0] == x[i] and result[j].c[1] ==
```

```
y[i] and result[j].c[2] == z[i] and result[j].c[3] == c[i]:
                    ax.text(x[i], y[i], z[i], f"{x[i]}, {y[i]},
{z[i]}, {c[i]}", color='k')
        fig.colorbar(img)

    if d <= 4:
        plt.show()
    else :
        print("\n*Tidak bisa plot dimensi lebih dari 4")
```

Function/Method	Keterangan
plotPoint()	Memvisualisasikan jarak titik terdekat untuk dimensi 1 – 4

File globals.py

```
import math
def initialize(dim: int):
    global count
    count = 0
    global max_point
    # max point = 1/c * 4 ^ d
    # c is constant in ball volume
    c = (math.sqrt(math.pi**dim) / math.gamma(dim/2 + 1))
    max_point = math.ceil((1/c) * 4 ** dim)
```

Function/Method	Keterangan
initialize(dim: int)	Inisiasi nilai count untuk menghitung banyak operasi yang dipakai dan nilai max_point yaitu banyak maksimum poin yang perlu dibandingkan

File solve.py

```
from point import *
from sort import *
from main import *
```

```
import globals

def euclidean_distance(p1: Point, p2: Point) -> float:
    globals.count += 1
    return math.sqrt(sum([(p1.c[i] - p2.c[i])**2 for i in
range(p1.d)]))

def check_coord(points: list, i: int, j:int, dist: int) -> bool:
    for d in range (points[0].d):
        if abs(points[i].c[d] - points[j].c[d]) > dist:
            return True
    return False

def solve_bruteForce(points: list):
    n = len(points)
    min = math.inf
    for i in range(n):
        for j in range(i+1, n):
            d = euclidean_distance(points[i], points[j])
            if d < min:
                min = d
                result = (points[i], points[j])
    return result, min

def solve(points: list, sort: int) -> tuple:
    n = len(points)
    dim = points[0].d

    if n == 1:
        return None, math.inf
    if n < 3:
        return solve_bruteForce(points)

    # Sort points berdasarkan x
    if sort == 1:
        points = merge_sort(points, 0)
    elif sort == 2:
        points = quicksort(points, 0)
    else:
```

```
        points.sort(key=lambda p: p.c[0])


    # Bagi menjadi dua bagian
    mid = n // 2
    mid_point = points[mid]
    left = points[:mid]
    right = points[mid:]

    # Cari pasangan terdekat di setiap bagian
    leftRes, leftDistance = solve(left, sort)
    rightRes, rightDistance = solve(right, sort)

    resDist = min(leftDistance, rightDistance)

    if  rightDistance > leftDistance:
        res = leftRes
    else:
        res = rightRes


    # Cari pasangan terdekat yang berada di strip
    strip = []
    for point in left:
        if point.c[0] >= mid_point.c[0] - resDist:
            strip.append(point)
    for point in right:
        if point.c[0] <= mid_point.c[0] + resDist:
            strip.append(point)

    # Sort strip berdasarkan y
    if (len(strip) > 1 and dim > 1):
        if sort == 1:
            strip = merge_sort(strip, 1)
        elif sort == 2:
            strip = quicksort(strip, 1)
        else:
            strip.sort(key=lambda p: p.c[1])
```

```
    for i in range (len(strip)):
        for j in range(i + 1, min(i + globals.max_point + 1,
len(strip))):
            if check_coord(strip, i, j, resDist):
                break
            mind = euclidean_distance(strip[i], strip[j])
            if mind < resDist:
                res = (strip[i], strip[j])
                resDist = mind

    return res, resDist
```

Function/Methode	Keterangan
euclidean_distance(p1: Point, p2: Point) -> float	Menghitung nilai euclidean _distance dari dua buah point.
check_coord(points: list, i: int, j:int, dist: int) -> bool	Menentukan apakah jarak tiap koordinat dari point ke-i dan poin ke-j lebih besar dari dist.
solve_bruteForce(points: list) -> tuple:	Menghitung jarak poin terdekat serta apa saja poinnya dengan algoritma bruteforce.
solve(points: list, sort: int) -> tuple	Menghitung jarak poin terdekat serta apa saja poinnya dengan algoritma divide and conquer.

File main.py

```
from point import *
from solve import *
from plotPoint import *
import globals
import time

def main():
    d = menu(1)
    n_point = menu(2)
```

```
    rounding = menu(3)
    # 1 Merge sort , 2 Quick sort
    sort = menu(4)

    bound = 10**3

    points = random_point(n_point , d, rounding, bound)


print("============================================================
======================")

    nn_dnc , nnDist_dnc = nearest_point(points, d, "dnc", sort)

    print()

    nn_b , nnDist_b = nearest_point(points, d, "brute")


print("============================================================
======================")
    visual = input("Do you want to visualize the result? (y/n): ")
    if visual == "y":
        plotPoint(points, nn_dnc, d)

def nearest_point(points: list, dim:int, key: str, sort:int = 1)
-> tuple:
    globals.initialize(dim)
    if key == "dnc":
        print(f"Result with divide and conquer:")
    elif key == "brute":
        print(f"Result with brute force:")
    start_time = time.time()
    if key == "dnc":
        nn, nnDist = solve(points, sort)
    elif key == "brute":
        nn, nnDist = solve_bruteForce(points)
    end_time = time.time()
    diff = end_time - start_time
```

```
    print(f"{diff * 10**3:0.9f} ms")
    print_points(nn)
    print(f"Distance: {nnDist:0.9f}")
    print(f"Number of euclidian operation: {globals.count}")
    return nn, nnDist

def menu(type : int) -> int:
    if type == 1:
        prompt = "Enter the dimension of the points: "
        minval = 2
        err = "Dimension must be greater than 0"
    elif type == 2:
        prompt = "Enter the number of points: "
        minval = 1
        err = "Number of points must be greater than 0"
    elif type == 3:
        prompt = "Enter the rounding of the points(ketelitian
angka dibelakang koma): "
        minval = 0
        err = "Rounding must be greater than 0"
    elif type == 4:
        prompt = "Enter the sorting method\n1. Merge sort(default)
\n2. Quick sort\n :"
        minval = 1
        err = "Sorting method must be 1 or 2"
    while True:
            try:
                num = int(input(prompt))
                if num < minval:
                    print(err)
                    continue
                break
            except ValueError:
                print("Invalid input")
                continue
    return num

def animation():
    animation = [
```

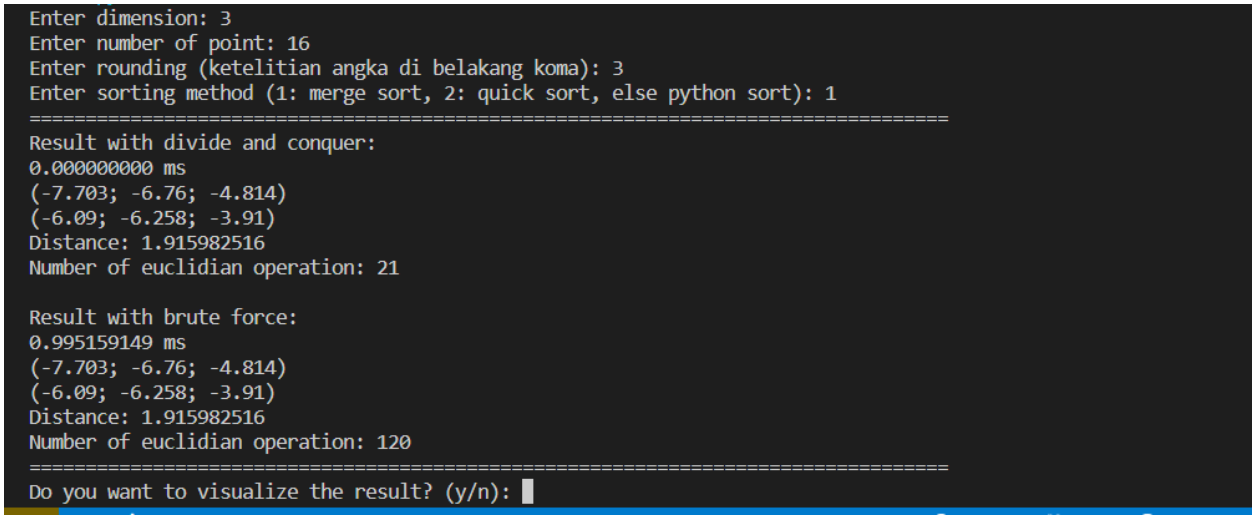
```
    "[         ]",
    "[=        ]",
    "[==       ]",
    "[===      ]",
    "[====     ]",
    "[=====    ]",
    "[======   ]",
    "[======= ]",
    "[========]",
    ]
    i = 0
    for i in range(len(animation)):
        print(animation[i % len(animation)], end='\r')
        time.sleep(0.1)

def startscreen():
    animation()
    print("Welcome to the nearest pair point program")
    input("Press enter to continue...")

if __name__ == "__main__":
    startscreen()
    main()
```

Function/Method	Keterangan
nearest_point(points: list, key: str, sort: int = 1) -> tuple:	Menghitung jarak terdekat dari sebuah list of points serta pasangan point dengan algoritma dan sorting tertentu.
menu(type : int) -> int	Memberikan prompt kepada user dan mengembalikan input dari user
animation()	Menjalankan animasi loading bar sederhana
startscreen()	Menjalankan prosedur animasi dan memberikan sambutan kepada user
main()	Menjalankan program utama

4. Test Case

Ket : limit random = 10

N = 16



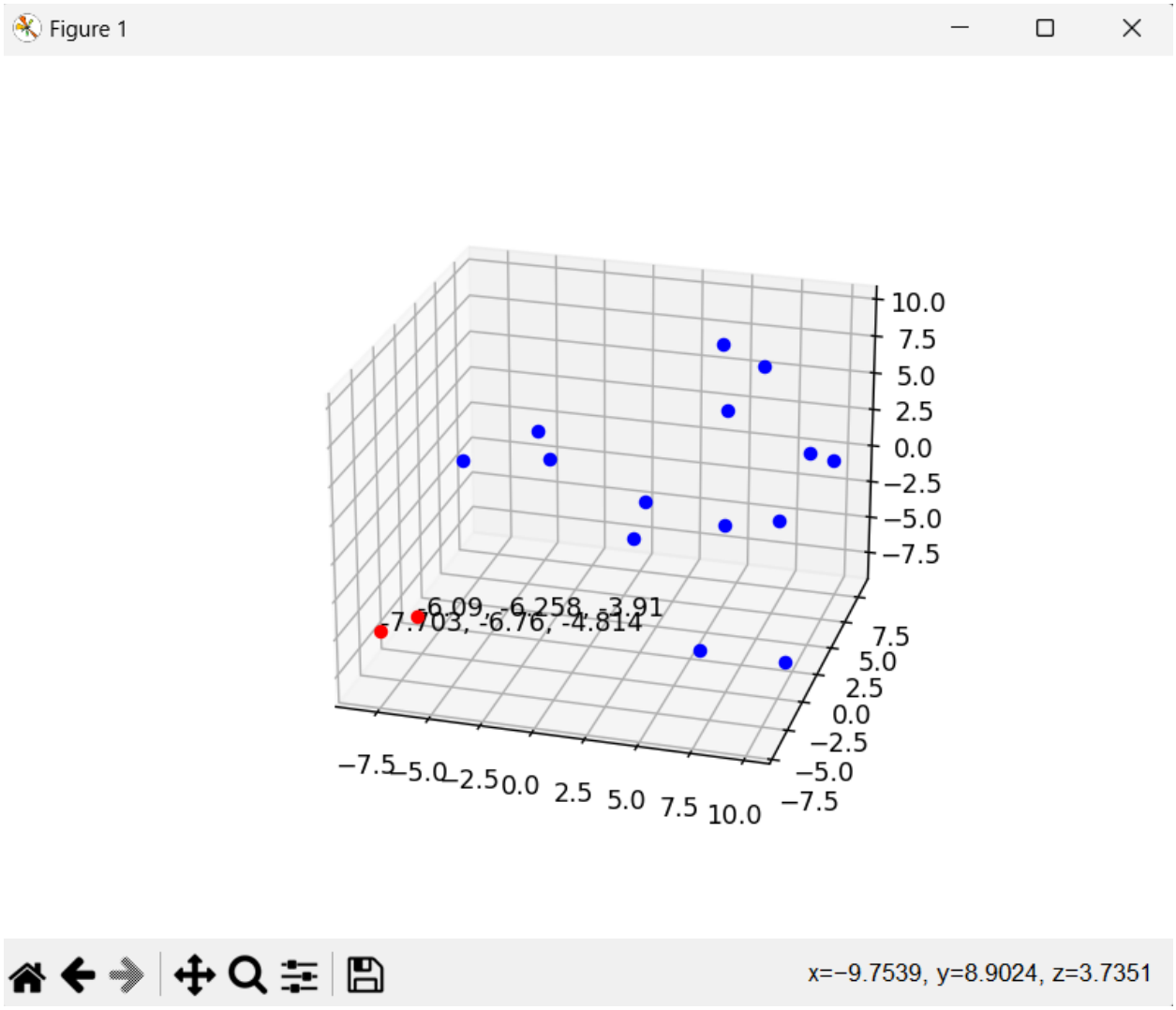


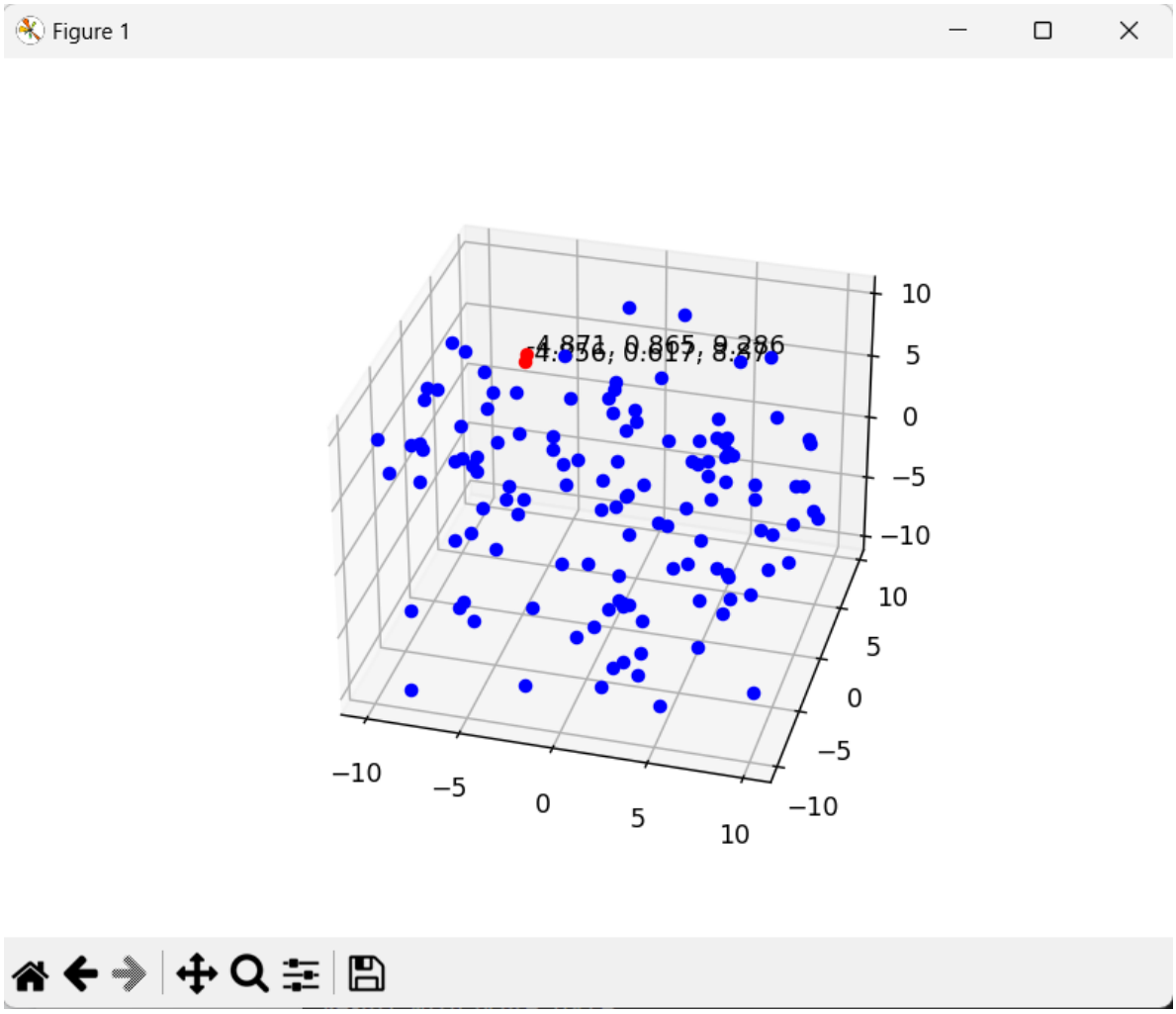
Figure 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
6.09, -6.258, -3.91
7.703, -6.76, -4.814
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x=−9.7539, y=8.9024, z=3.7351

N = 64

```
Enter dimension: 3
Enter number of point: 64
Enter rounding (ketelitian angka di belakang koma): 3
Enter sorting method (1: merge sort, 2: quick sort, else python sort): 1
================================================================================
Result with divide and conquer:
2.566337585 ms
(2.265; 7.145; -6.003)
(1.833; 8.126; -5.767)
Distance: 1.097579610
Number of euclidian operation: 75

Result with brute force:
6.551265717 ms
(2.265; 7.145; -6.003)
(1.833; 8.126; -5.767)
Distance: 1.097579610
Number of euclidian operation: 2016
================================================================================
Do you want to visualize the result? (y/n): y
```

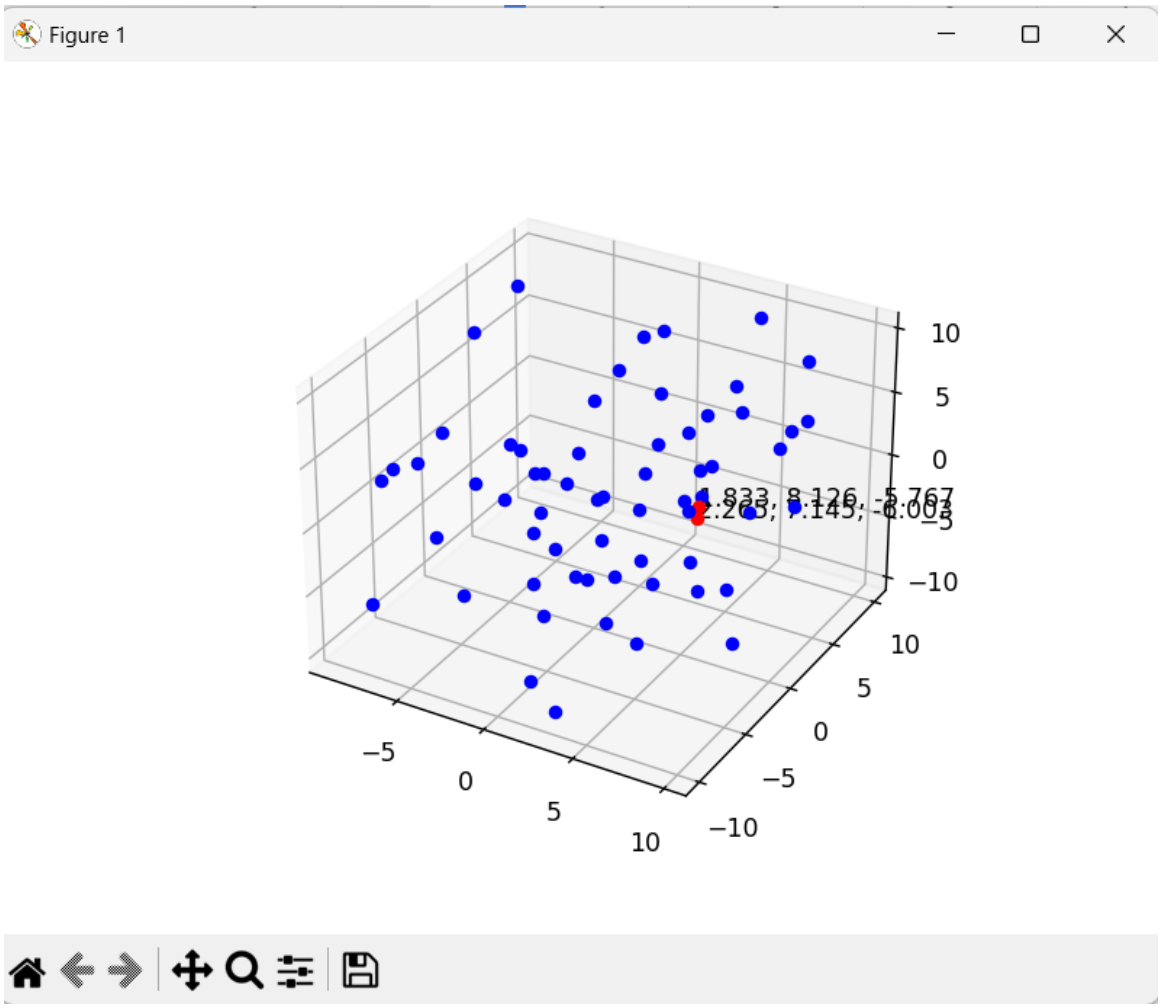
N = 128

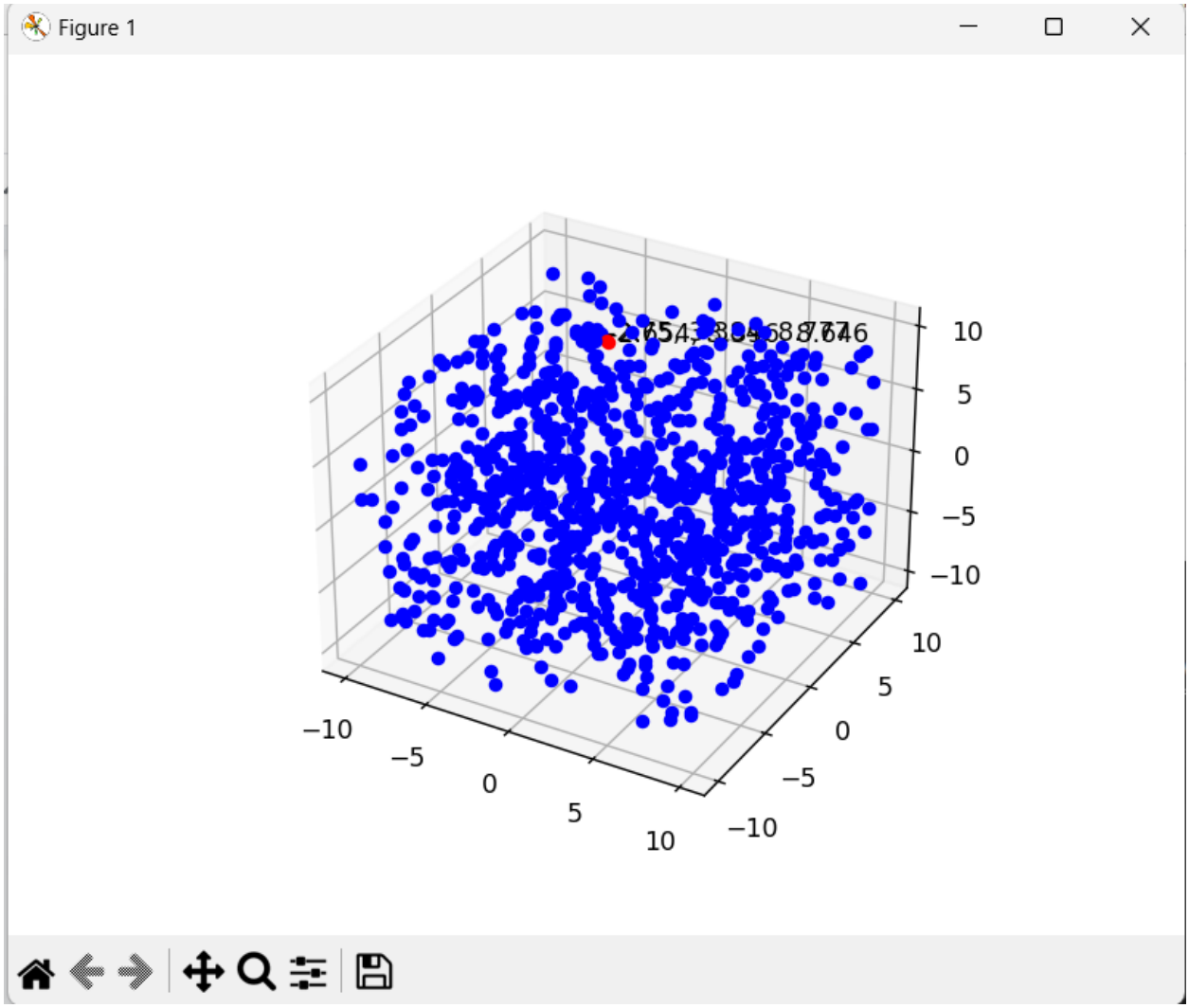
```
Enter dimension: 3
Enter number of point: 128
Enter rounding (ketelitian angka di belakang koma): 3
Enter sorting method (1: merge sort, 2: quick sort, else python sort): 1
================================================================================
Result with divide and conquer:
7.006406784 ms
(-4.856; 0.617; 8.87)
(-4.871; 0.865; 9.286)
Distance: 0.484546179
Number of euclidian operation: 130

Result with brute force:
27.536869049 ms
(-4.871; 0.865; 9.286)
(-4.856; 0.617; 8.87)
Distance: 0.484546179
Number of euclidian operation: 8128
================================================================================
Do you want to visualize the result? (y/n): y
```



N = 1000

```
Enter dimension: 3
Enter number of point: 1000
Enter rounding (ketelitian angka di belakang koma): 3
Enter sorting method (1: merge sort, 2: quick sort, else python sort): 1
================================================================================
Result with divide and conquer:
70.810317993 ms
(-2.75; 3.384; 8.777)
(-2.654; 3.396; 8.646)
Distance: 0.162852694
Number of euclidian operation: 1159

Result with brute force:
1767.072439194 ms
(-2.654; 3.396; 8.646)
(-2.75; 3.384; 8.777)
Distance: 0.162852694
Number of euclidian operation: 499500
================================================================================
Do you want to visualize the result? (y/n): y
```





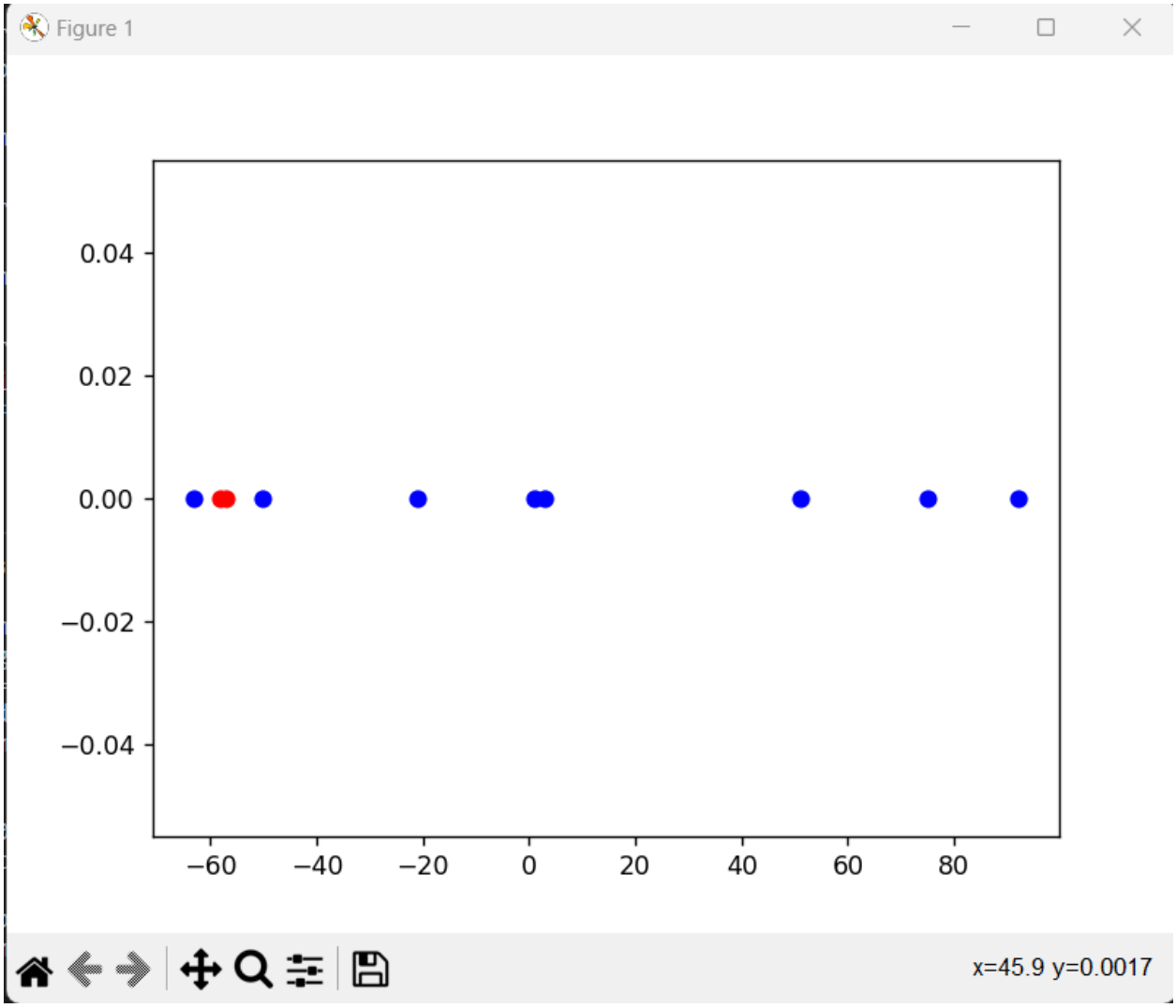
Dari 4 pengujian di atas, dengan menggunakan limit random (-10,10) dan rounding (3) didapatkan perbandingan sebagai berikut.

N	Result Pair	Distance	Execution Time(ms)		Total Euclidian Ops	
			Dnc	BF	DnC	BF
16	(-7.703; -6.76; -4.814) (-6.09; -6.258; -3.91)	1.915	0.00	0.995	21	120
64	(2.265; 7.145; -6.003) (1.833; 8.126; -5.767)	1.098	2.566	6.551	75	2016
128	(-4.856; 0.617; 8.87) (-4.871; 0.865; 9.286)	0.485	7.006	27.537	130	8128

1000	(-2.75; 3.384; 8.777) (-2.654; 3.396; 8.646)	0.162	70.810	1767.072	1159	499500

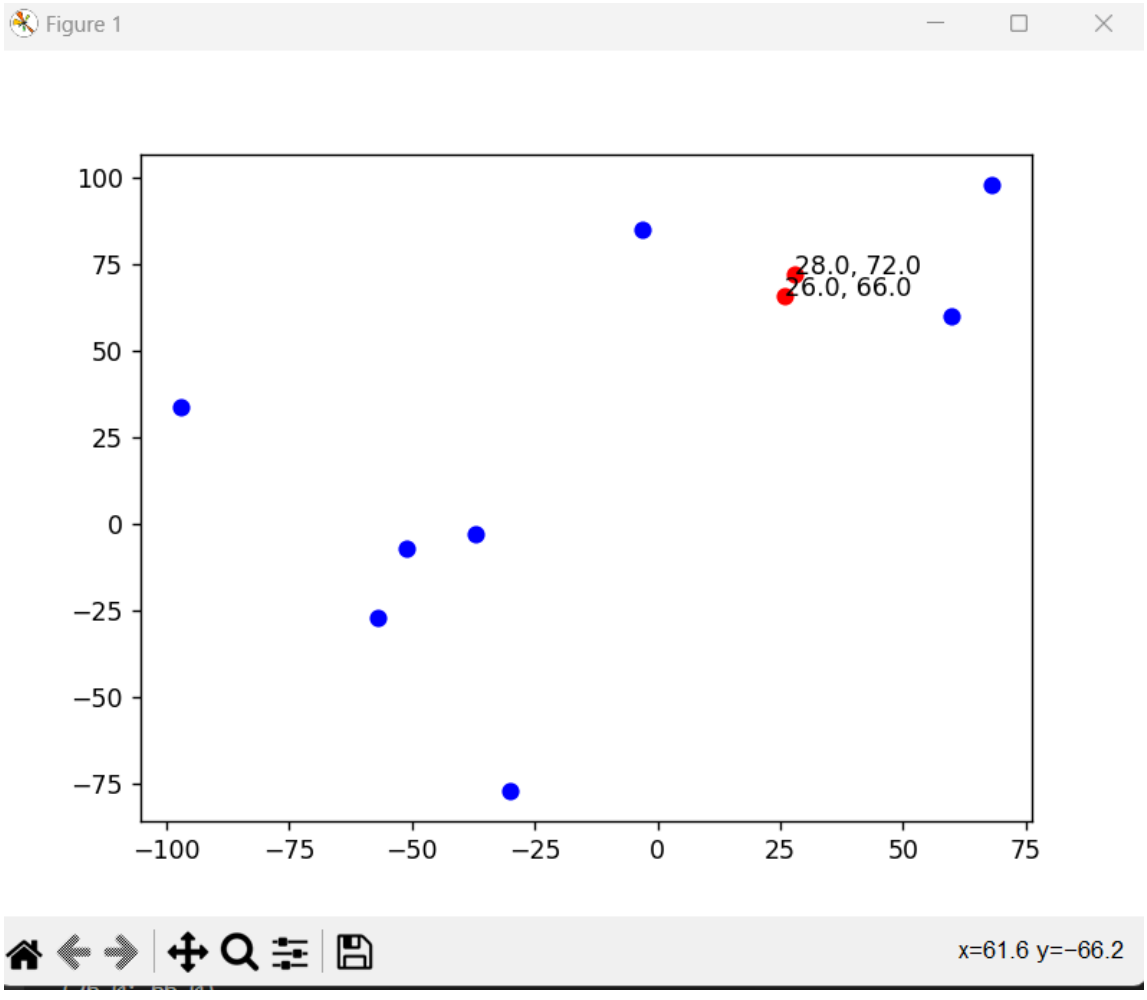
Dimensi 1

```
Result with divide and conquer:
1.016139984 ms
(-58.0)
(-57.0)
Distance: 1.000000000
Number of euclidian operation: 7

Result with brute force:
0.000000000 ms
(-57.0)
(-58.0)
Distance: 1.000000000
Number of euclidian operation: 45
================================================================================
Do you want to visualize the result? (y/n): y
```



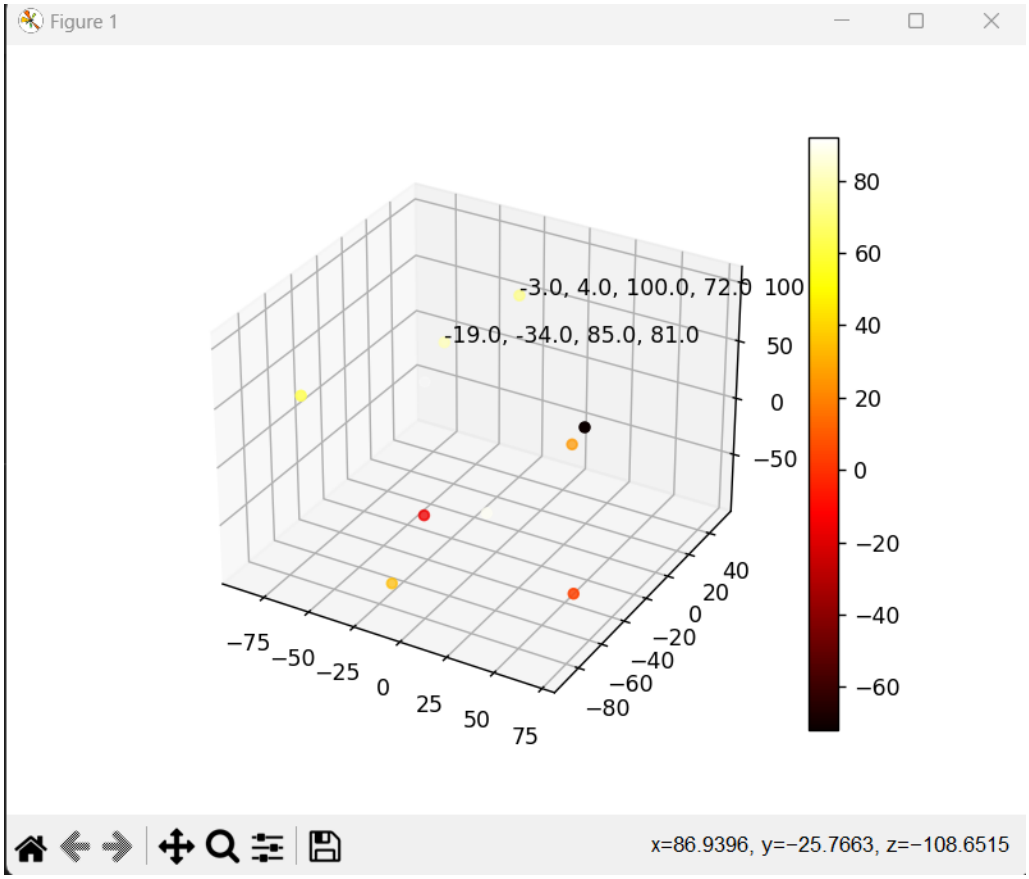
Dimensi 2

```
Enter dimension: 2
Enter number of point: 10
Enter rounding (ketelitian angka di belakang koma): 0
Enter sorting method (1: merge sort, 2: quick sort, else python sort): 1
================================================================================
Result with divide and conquer:
0.000000000 ms
(26.0; 66.0)
(28.0; 72.0)
Distance: 6.324555320
Number of euclidian operation: 11

Result with brute force:
0.000000000 ms
(28.0; 72.0)
(26.0; 66.0)
Distance: 6.324555320
Number of euclidian operation: 45
================================================================================
Do you want to visualize the result? (y/n): y
```



Dimensi 4

```
Enter dimension: 4
Enter number of point: 10
Enter rounding (ketelitian angka di belakang koma): 0
Enter sorting method (1: merge sort, 2: quick sort, else python sort): 1
================================================================================
Result with divide and conquer:
0.000000000 ms
(-19.0; -34.0; 85.0; 81.0)
(-3.0; 4.0; 100.0; 72.0)
Distance: 44.788391353
Number of euclidian operation: 13

Result with brute force:
0.000000000 ms
(-19.0; -34.0; 85.0; 81.0)
(-3.0; 4.0; 100.0; 72.0)
Distance: 44.788391353
Number of euclidian operation: 45
================================================================================
Do you want to visualize the result? (y/n): y
```



Keterangan. Pada visualisasi dimensi-4, program menggunakan *heat map* sebagai indikator dimensi ke-4. Pasangan titik yang paling terdekat ditandai dengan adanya keterangan posisi pada visualisasi.

Dimensi Lainnya

```
Enter dimension: 6
Enter number of point: 10
Enter rounding (ketelitian angka di belakang koma): 0
Enter sorting method (1: merge sort, 2: quick sort, else python sort): 1
================================================================================
Result with divide and conquer:
0.000000000 ms
(34.0; -71.0; -31.0; -29.0; 32.0; -24.0)
(-0.0; -58.0; -42.0; 23.0; -18.0; 32.0)
Distance: 98.924213416
Number of euclidian operation: 27

Result with brute force:
0.000000000 ms
(34.0; -71.0; -31.0; -29.0; 32.0; -24.0)
(-0.0; -58.0; -42.0; 23.0; -18.0; 32.0)
Distance: 98.924213416
Number of euclidian operation: 45
================================================================================
Do you want to visualize the result? (y/n): y

*Tidak bisa plot dimensi lebih dari 4
```

```
Enter dimension: 10
Enter number of point: 100
Enter rounding (ketelitian angka di belakang koma): 0
Enter sorting method (1: merge sort, 2: quick sort, else python sort): 1
================================================================================
Result with divide and conquer:
4.111528397 ms
(22.0; -83.0; -99.0; -80.0; 52.0; -46.0; 48.0; -54.0; -37.0; -54.0)
(-47.0; -80.0; -70.0; -70.0; 89.0; -43.0; 38.0; -88.0; -3.0; -48.0)
Distance: 97.657564991
Number of euclidian operation: 428

Result with brute force:
11.145353317 ms
(-47.0; -80.0; -70.0; -70.0; 89.0; -43.0; 38.0; -88.0; -3.0; -48.0)
(22.0; -83.0; -99.0; -80.0; 52.0; -46.0; 48.0; -54.0; -37.0; -54.0)
Distance: 97.657564991
Number of euclidian operation: 4950
================================================================================
Do you want to visualize the result? (y/n): y

*Tidak bisa plot dimensi lebih dari 4
```

Akan tetapi, dimensi 4 dan selanjutnya algoritma divide and conquer masih menghasilkan jawaban yang berbeda dengan hasil algoritma brute force. Seperti berikut

```
Enter the dimension of the points: 4
Enter the number of points: 1000
Enter the rounding of the points(ketelitian angka dibelakang koma): 3
Enter the sorting method
1. Merge sort(default)
2. Quick sort
: 1
================================================================================
Result with divide and conquer:
28.233051300 ms
(-422.314; 709.795; -657.507; -532.562)
(-407.949; 731.312; -642.757; -594.841)
Distance: 69.033114192
Number of euclidian operation: 1245

Result with brute force:
673.892498016 ms
(84.033; 328.05; 422.876; -169.382)
(52.787; 345.684; 373.624; -156.53)
Distance: 62.275226856
Number of euclidian operation: 499500
================================================================================
```

```
Enter the dimension of the points: 10
Enter the number of points: 1000
Enter the rounding of the points(ketelitian angka dibelakang koma): 3
Enter the sorting method
1. Merge sort(default)
2. Quick sort
: 1
================================================================================
Result with divide and conquer:
48.218727112 ms
(408.852; -720.03; -4.298; -758.377; -65.521; -263.733; 491.882; -558.49; 553.481; -15.078)
(464.86; -665.009; 171.705; -976.447; -195.79; -214.416; 138.633; -570.148; 579.718; -226.111)
Distance: 523.681237478
Number of euclidian operation: 4120

Result with brute force:
1192.425251007 ms
(-897.941; 911.526; -477.218; 594.057; 211.713; -129.479; 1.167; 984.315; 817.871; -136.352)
(-714.071; 794.635; -329.932; 650.305; 325.553; -52.288; -77.558; 827.219; 705.875; -153.056)
Distance: 367.349409466
Number of euclidian operation: 499500
================================================================================
```

Hal tersebut terjadi karena pada algoritma masih belum mengatasi semua kasus pada dimensi tinggi khususnya kasus penyelesaian yang seharusnya diselesaikan di satu dimensi lebih rendah.

5. Daftar Pustaka

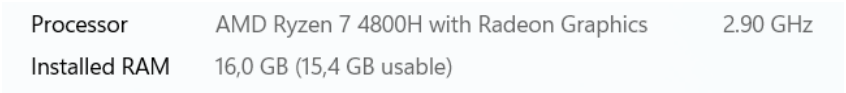
Munir, Rinaldi. 2023. IF2211 Strategi Algoritma https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf, diakses 25 Februari 2023

Suri , Subhash. Closest Pair Problem, https://sites.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf, diakses 27 Februari 2023

6. Lampiran

Github : https://github.com/yansans/Tucil2_13521110_13521120

Spesifikasi Laptop untuk Test Case:



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