PEMANFAATAN ALGORITMA *DIVIDE AND CONQUER* DALAM PENCARIAN *NEAREST POINTS*

Diajukan sebagai pemenuhan tugas kecil II.

Oleh:

1. 13521111 - Tabitha Permalla
2. 13521139 - Nathania Calista

Dosen Pengampu : Dr. Ir. Rinaldi, M.T

IF2211 - Strategi Algoritma

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

BAB 1
DESKRIPSI MASALAH

Permasalahan yang dibahas pada tugas kali ini adalah menemukan *nearest points* atau biasa dikenal juga dengan *K-Nearest Neighbor* (KNN). Dalam permasalahan ini, diberikan x buah titik dalam ruang n dimensi. Setiap titik di dalam ruang dinyatakan dengan P = (x1, x2, x3,..., xn), dengan n merupakan jumlah dimensi dalam ruang. Tujuan akhir dari permasalahan ini adalah menemukan sepasang titik yang memiliki jarak terdekat jika dibandingkan dengan titik - titik lain. Jarak antara dua buah titik P_1 = (X_{11},X_{12},....X_{1n}) dan P2 = (X_{21}, X_{22},....,X_{2n}) dapat dihitung dengan menggunakan rumus Euclidean berikut :

$$d = \sqrt{(x_{21} - x_{11})^2 + (x_{22} - x_{12})^2 + \ldots\ldots\ldots + (x_{2n} - x_{1n})^2}$$

Algoritma yang dipakai dalam memecahkan masalah ini adalah algoritma *divide and conquer* dan algoritma *brute force* dengan tujuan membandingkan kedua algoritma tersebut dan mendapatkan algoritma yang paling efektif

BAB 2
TEORI DASAR

2.1. Algoritma *Divide and Conquer*

Divide and Conquer dulunya adalah strategi militer yang dikenal dengan nama *divide ut imperes*. Sekarang ini, *divide and conquer* menjadi salah satu strategi fundamental dalam ilmu komputer dengan nama algoritma *divide and conquer*.

Algoritma *divide and conquer* terdiri dari 3 bagian, yaitu *divide, conquer (solve),* dan *combine*. Pada bagian *divide,* persoalan dibagi menjadi menjadi beberapa sub-persoalan yang memiliki kemiripan dengan soal semula namun berukuran lebih kecil (hampir sama). Pada bagian *conquer*, algoritma menyelesaikan masing - masing upa persoalan secara langsung atau secara rekursif. Biasanya, penyelesaian secara langsung hanya akan terjadi jika upa-persoalan sudah berukuran kecil. Namun, jika upa-persoalan masih berukuran terlalu besar, penyelesaian akan dilakukan secara rekursif. Pada bagian terakhir, *combine*, solusi masing - masing upa-persoalan akan digabungkan dan membentuk solusi persoalan semula.

Algoritma ini memiliki kompleksitas algoritma yang lebih baik jika dibandingkan dengan algoritma *brute force*. Beberapa contoh persoalan yang dapat diselesaikan dengan algoritma ini adalah persoalan MinMaks, menghitung perpangkatan, pengurutan (*sorting*), mencari pasangan titik terdekat, *Convex Hull,* perkalian matriks, dan masih banyak lagi. Pada laporan ini, persoalan *sorting* dan pencarian pasangan titik terdekat akan dibahas dengan lebih *detail.*

2.2. Algoritma *Brute Force*

Algoritma brute force adalah pendekatan dari penyelesaian sebuah masalah algoritmik dengan menggunakan pendekatan yang lempeng atau straight-forward. Algoritma ini didasarkan pada pernyataan pada persoalan (problem statement) dan definisi konsep yang dilibatkan. Algoritma ini memecahkan persoalan dengan sangat sederhana, langsung,dan jelas.

Karakteristik dari algoritma brute force adalah tidak cerdas, karena membutuhkan jumlah komputasi yang besar dan waktu yang lama. Selain itu, algoritma brute force lebih cocok untuk persoalan yang berukuran kecil karena algoritma ini sederhana dan implementasinya mudah. Meskipun bukan algoritma tercepat, algoritma ini dapat menyelesaikan hampir semua persoalan.

2.3. Sorting

Algoritma *divide and conquer* dapat digunakan untuk mengurutkan data - data yang ada. Ide mengurutkan larik dengan algoritma *divide and conquer* adalah sebagai berikut :

1. Jika ukuran larik = 1, larik sudah terurut dengan sendirinya
2. Jika ukuran larik > 1, bagi larik menjadi dua bagian dan susun masing - masing bagian
3. Gabungkan solusi dari masing - masing bagian menjadi satu solusi utuh

Terdapat dua pendekatan melakukan pengurutan dengan *divide and conquer,* yaitu:

- **Mudah membagi, tetapi sulit menggabung (*easy split/hard join*)**
  Pembagian larik menjadi dua bagian mudah secara komputasi, karena hanya membagi berdasarkan indeks. Penggabungan dua buah larik terurut menjadi sebuah larik terurut sukar secara komputasi (kompleksitas algoritmanya tinggi)
- **Sulit membagi, tapi mudah menggabung (*hard split / easy join*)**
  Pembagian larik menjadi dua bagian sukar secara komputasi karena pembagiannya berdasarkan nilai elemen, bukan posisi elemen larik. Penggabungan dua buah larik terurut menjadi sebuah larik terurut mudah secara komputasi

2.4. *Merge Sort*

Algoritma *merge sort* akan mengurutkan larik dengan metode *divide and conquer.* Jika diberikan sebuah larik A dengan panjang n, maka algoritma *merge sort* yang digunakan adalah sebagai berikut :

1. Jika n = 1, maka larik A sudah terurut dengan sendirinya (langkah SOLVE)
2. Jika n > 1 maka :
   a. DIVIDE
      Larik A dibagi menjadi dua bagian pada posisi pertengahan, masing - masing beukuran n/2 elemen
   b. CONQUER
      Secara rekursif, terapkan *Merge Sort* pada masing - masing bagian
   c. MERGE

Gabungkan hasil pengurutan kedua bagian sehingga diperoleh larik A yang terurut.

Kompleksitas algoritma *Merge Sort* diukur dari jumlah operasi perbandingan elemen - elemen larik.Kompleksitas algoritma dari seluruh proses *merge sort* adalah sekitar $O(n\ ^{2}\log_{n})$, lebih baik daripada kompleksitas algoritma pengurutan secara *brute force*.

BAB 3
ALGORITMA *DIVIDE AND CONQUER*

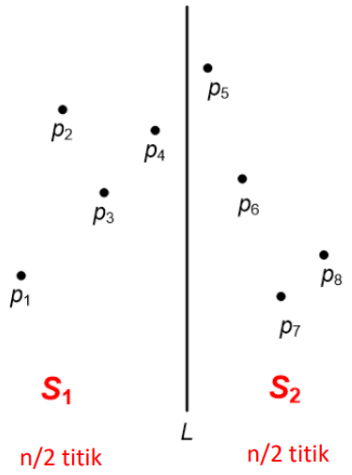
Secara umum, algoritma *divide and conquer* dalam pemecahan masalah pencarian titik terdekat dapat dibagi ke beberapa langkah sebagai berikut :

1. **Urutkan seluruh titik**

   Dalam program ini, titik yang digunakan adalah titik dengan koordinat yang dihasilkan secara *random*. Seluruh titik yang dihasilkan nantinya akan diurutkan berdasarkan nilai koordinat X. Algoritma yang digunakan dalam pengurutan titik - titik ini adalah *merge sort*.

2. ***DIVIDE***

   Jika banyaknya titik tidak sama dengan 2, bagi titik - titik yang ada ke dalam dua bagian yang sama banyak. Agar dapat memudahkan penjelasan, kita misalkan kedua titik terbagi ke dalam dua sisi, yaitu S1 dan S2. Gambaran lebih jelas dapat dilihat pada gambar 3.1 di bawah ini



Gambar 3.1 Membagi titik ke dalam dua bagian sama banyak

3. ***SOLVE***

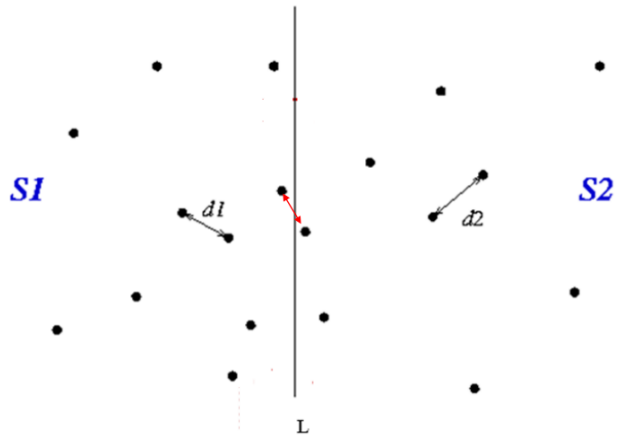
   Jika banyak titik sudah sama dengan 2, kembalikan sepasang titik tersebut.

4. ***CONQUER***

   Jika banyak titik lebih besar sama dengan 2, terapkan fungsi rekursif yang terus membagi titik - titik menjadi 2 kelompok yang sama besar.

5. COMBINE

Pada langkah ini, algoritma akan menggabungkan solusi yang dihasilkan dari larik bagian kiri dengan larik bagian kanan. Namun, tidak hanya 2 jawaban yang akan dibandingkan, melainkan 3 jawaban. Karena, selain menghitung jarak titik - titik yang sudah dibagi 2, algoritma ini juga menghitung kasus dimana ada titik - titik yang berada dekat dengan “perbatasan” atau dapat dilihat pada gambar 3.2 di bawah ini.



Gambar 3.2 Menghitung titik di perbatasan

Sebelum membandingkan titik - titik di area perbatasan, algoritma yang kami buat membandingkan terlebih dahulu jarak dari pasangan terdekat sisi kiri dengan jarak dari pasangan terdekat sisi kanan. Pasangan yang memiliki jarak terdekat akan disimpan sebagai pasangan titik terdekat dan jaraknya akan disimpan ke dalam sebuah variabel *temporary* yang kami namakan sebagai variabel *min.*

Cara menghitung jarak dari titik - titik yang berada pada garis perbatasan (garis L pada gambar 3.2) diawali dengan menyimpan seluruh titik - titik di sekitar garis L pada sebuah *array of points.* Syarat yang harus dipenuhi titik agar masuk ke dalam *array of points* adalah memiliki selisih absis maksimal sebesar variabel *min.* Selisih absis diperoleh dengan mengurangkan absis dari titik dengan absis garis L atau *center* (melambangkan perbatasan antara dua sisi).

Setelah mendapatkan semua titik - titik yang berada dekat dengan garis perbatasan, hitung jarak seluruh titik itu dengan satu sama lain, menggunakan algoritma *brute force*, yaitu dengan melakukan *looping* sebanyak jumlah titik di dalam *array of points*. Jika ada pasangan titik yang memiliki jarak lebih kecil dari variable *min*, berarti sepasang titik itu akan disimpan sebagai pasangan titik terdekat.

BAB 4
IMPLEMENTASI

Dalam membuat program, penulis menggunakan bahasa *Python,* dengan implementasi sebagai berikut

4.1. File brtuteforce.py

File ini berisi variasi untuk memecahkan permasalahan dengan algoritma *brute force* dan akan digunakan sebagai pembanding untuk membuktikan bahwa algoritma *divide and conquer* memiliki kompleksitas algoritma yang lebih baik. Kode yang digunakan dalam *file* ini dapat dilihat di bawah

```
import distance as d

def nearest_points(points):
    n = len(points)
    nearest2 = [points[0],points[1]]
    min2 = d.distance(points[0],points[1])
    count = 1
    for i in range(n):
        for j in range(i+1,n):
            count += 1
            if d.distance(points[i],points[j]) < min2:
                min2 = d.distance(points[i],points[j])
                nearest2 = [points[i],points[j]]
    return nearest2, count
```

4.2. File distance.py

File ini berisi algoritma utama dalam mencari pasangan titik terdekat dan berisi rumus untuk menghitung jarak antara dua buah titik pada n buah dimensi. Kode yang digunakan dalam *file* ini dapat dilihat di bawah

```
import math
import sort as s
```

```
# Rumus untuk menghitung jarak
def distance(p1,p2):
         return  math.sqrt(sum([(p1[i]  -  p2[i])**2  for  i  in
range(len(p1))]))

#Rumus mencari nearest points
def nearest_points(points):
    count = 0
    n = len(points)
    if (n <= 1):
        return None, count
    elif (n == 2):
        return points, count
    else:
        left = points[:(n//2)]
        right = points[(n//2):]

        nearest_left, countleft = nearest_points(left)
        nearest_right, countright = nearest_points(right)

        count = count + countleft + countright

        if (nearest_left is None):
            nearest = nearest_right
        elif (nearest_right is None):
            nearest = nearest_left
        else:
            left_d = distance(nearest_left[0],nearest_left[1])
            right_d = distance(nearest_right[0],nearest_right[1])
            count += 2

            if (right_d < left_d):
                min = right_d
                nearest = nearest_right
            else:
                min = left_d
                nearest = nearest_left
```

```
            center = (left[-1][0] + right[0][0]) / 2
               mid = [point for point in points if(abs(point[0] -
center) < min)]
            n_mid = len(mid)

            for i in range (n_mid):
                for j in range (i+1, n_mid):
                    d = distance(mid[i], mid[j])
                    count += 1
                    if (d < min):
                        nearest = [mid[i],mid[j]]
                        min = d
        return nearest, count
```

4.3. File input.py

File ini berisi algoritma untuk meminta *input* dari user dan memvalidasi *input* sampai mendapatkan *input* yang valid dan dapat dijalankan oleh program. Kode yang digunakan dalam file ini adalah sebagai berikut

```
def welcomeMsg():

print("=====================================================")
print("================  NEAREST  POINT  ====================")
print("=====================================================")

def inputUser():
    while(True):
        titik = int(input("Please input the number of points: "))
        if(titik > 1):
            break
        else :
             print("Input not valid, a minimum number of 2 points
is needed, please reenter your input!")

    while(True):
```

```
        dimensi = int(input("Please input the dimension: "))
        if(dimensi > 0):
            break
        else :
              print("Input not valid, a minimum of 1 dimension is
needed, please reenter your input!")
    return titik,dimensi
```

4.4. File main.py

File ini merupakan gabungan dari semua fungsi yang sudah dibuat dan siap diimplementasikan. Kode yang digunakan dalam file ini adalah sebagai berikut

```
import randomizer as r
import distance as d
import plot as p
import sort as s
import input as i
import bruteforce as b
import time as tm

i.welcomeMsg()
n,dim = i.inputUser()

points = r.randomizer(n,dim)
print("====================            DONE            RANDOMIZING
====================")
print("Do   you   want   to   see   all   the   points   that   were
generated?(y/n)")
print("============================================================
")
options=input()
while(True):
     if(options == "y" or options == "Y" or options == "N" or
options == "n"):
        break
    else:
```

```
        print("Input not valid!")

if(options == "y" or options == "Y"):
    for i in range(n):
        print(points[i])

# Divide and Conquer
sorted = s.merge_sort(points)
st = tm.perf_counter_ns()
nearest, count = d.nearest_points(sorted)
et = tm.perf_counter_ns()

print("\033[91m================  Divide  and  Conquer  Solution
=============== \033[0m ")
print("Your nearest points are : ")
print(nearest[0])
print(nearest[1])
print("Distance : ", d.distance(nearest[0],nearest[1]))
print("Euclidean distance computation count : ", count)
print("\033[92mExecution          time          :          {0:.2f}
".format((et-st)/1000000),"miliseconds\033[0m")
print("\033[91m================================================
========== \033[0m")
print()

p.plot(dim,points,nearest)

# Brute Force
st2 = tm.perf_counter_ns()
nearest2, count2 = b.nearest_points(points)
et2 = tm.perf_counter_ns()

print("\033[93m==================    Brute    Force    Solution
==================  \033[0m")
print("Your nearest points are : ")
print(nearest2[0])
print(nearest2[1])
print("Distance : ", d.distance(nearest2[0],nearest2[1]))
print("Euclidean distance computation count : ", count2)
print("\033[92mExecution          time          :          {0:.2f}
".format((et2-st2)/1000000),"miliseconds\033[0m ")
```

```
print("\033[93m==============================================
============  \033[0m")
```

4.5. File plot.py

File ini berisi kode untuk membuat visualisasi titik - titik yang ada beserta pasangan titik terdekat. Pasangan titik terdekat akan berwarna merah, sedangkan titik - titik lain akan berwarna biru. Kode yang digunakan dalam file ini adalah sebagai berikut

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

def plot(dim, points, nearest):
    others = []
    for p in points:
        if p not in nearest:
            others.append(p)

    if dim == 1:
        fig, ax = plt.subplots()

        x = [p[0] for p in others]
        ax.plot(x, [0]*len(others), 'bo')

        if nearest is not None:
            n = [p[0] for p in nearest]
            ax.plot(n, [0,0], 'ro')

        ax.set_yticks([])

        ax.axhline(y=0, color='k')


    elif dim == 2:
```

```
        x = [p[0] for p in others]
        y = [p[1] for p in others]

        plt.plot(x, y, 'bo')

        if nearest is not None:
            p1,p2 = nearest
            plt.plot([p1[0],p2[0]],[p1[1],p2[1]],'ro')

    elif dim == 3:
        plot = plt.figure()
        axis = plot.add_subplot(111, projection='3d')

        x = [p[0] for p in others]
        y = [p[1] for p in others]
        z = [p[2] for p in others]
        axis.scatter(x, y, z, c='b')

        # Plot the nearest pair of points in red.
        if nearest is not None:
            p1, p2 = nearest
              axis.scatter([p1[0], p2[0]], [p1[1], p2[1]], [p1[2],
p2[2]], c='r')

        axis.set_xlabel('X')
        axis.set_ylabel('Y')
        axis.set_zlabel('Z')

    else:
        print("Dimension is not visualizable")

    plt.show()
```

4.6. File randomizer.py

File ini berisi kode yang digunakan oleh program untuk menghasilkan koordinat *random*. Kode yang digunakan dalam file ini adalah sebagai berikut

```
import random

def randomizer(n,dim):
    points = []
    for i in range(n):
        point = [random.uniform(-1000,1000) for _ in range(dim)]
        points.append(point)
    return points
```

4.7. File sort.py

File ini berisi kode untuk melakukan *sorting* dengan algoritma *merge sort*. Kode yang digunakan dalam file ini adalah sebagai berikut

```
def merge_sort(arr):
    if(len(arr) == 1):
        return arr
    else:
        middle = len(arr) //2
        left_arr = arr[:middle]
        right_arr = arr[middle:]

        left_arr = merge_sort(left_arr)
        right_arr = merge_sort(right_arr)
        return merge(left_arr,right_arr)

def merge(left_arr, right_arr):
    sorted_arr = []
    left_index = 0
```

```
    right_index = 0
      while (left_index < len(left_arr) and right_index <
len(right_arr)):
        if(left_arr[left_index] < right_arr[right_index]):
            sorted_arr.append(left_arr[left_index])
            left_index+=1
        else :
            sorted_arr.append(right_arr[right_index])
            right_index +=1

    sorted_arr += left_arr[left_index:]
    sorted_arr += right_arr[right_index:]
    return sorted_arr
```

BAB 5

Pengujian

5.1. Validasi Input

Dalam proses ini, dilakukan validasi terhadap input pengguna. Jumlah titik harus lebih besar sama dengan 2, sedangkan dimensi harus lebih besar sama dengan 1.

Bukti screenshot:

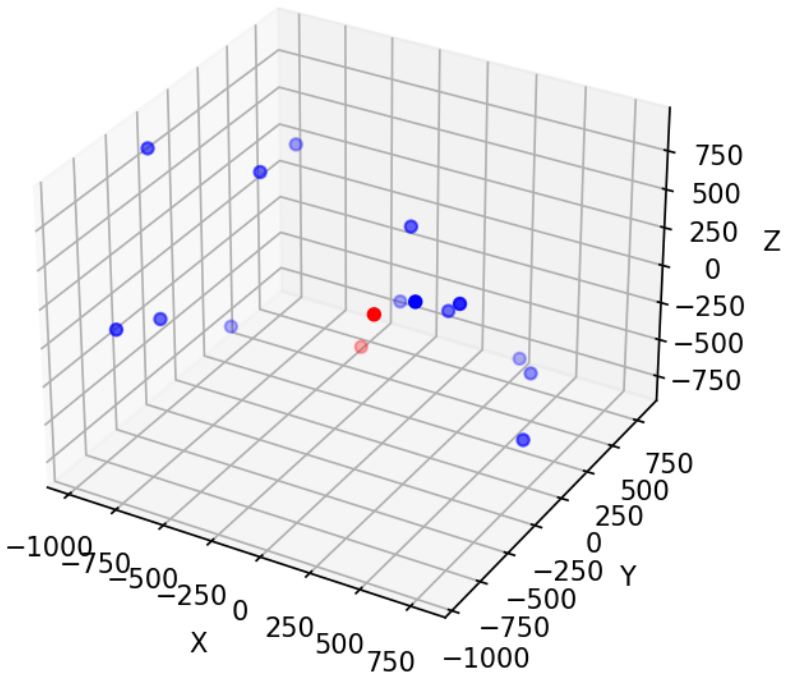
```
==================================================
=============== NEAREST POINT ====================
==================================================
Please input the number of points: 1
Input not valid, a minimum number of 2 points is needed, please reenter your input!
Please input the number of points: 0
Input not valid, a minimum number of 2 points is needed, please reenter your input!
Please input the number of points: -3
Input not valid, a minimum number of 2 points is needed, please reenter your input!
Please input the number of points: 7
Please input the dimension: 0
Input not valid, a minimum of 1 dimension is needed, please reenter your input!
Please input the dimension: -4
Input not valid, a minimum of 1 dimension is needed, please reenter your input!
Please input the dimension: 3
```

5.2. Test-case n = 16 (dimensi 3)

Bukti screenshoot:

```
=====================================================
================ NEAREST POINT ======================
=====================================================
Please input the number of points: 16
Please input the dimension: 3
==================== DONE RANDOMIZING ====================
Do you want to see all the points that were generated?(y/n)
=========================================================
n
================= Divide and Conquer Solution ================
Your nearest points are :
[180.02132553347087, -498.3472088476935, 133.32115814002304]
[244.75574535173132, -499.6360682525007, 365.52372120075006]
Distance :  241.060649131589
Euclidean distance computation count :  79
Execution time : 0.32  miliseconds
=============================================================

==================== Brute Force Solution ===================
Your nearest points are :
[244.75574535173132, -499.6360682525007, 365.52372120075006]
[180.02132553347087, -498.3472088476935, 133.32115814002304]
Distance :  241.060649131589
Euclidean distance computation count :  121
Execution time : 0.75  miliseconds
=============================================================
```

Hasil visualisasi:

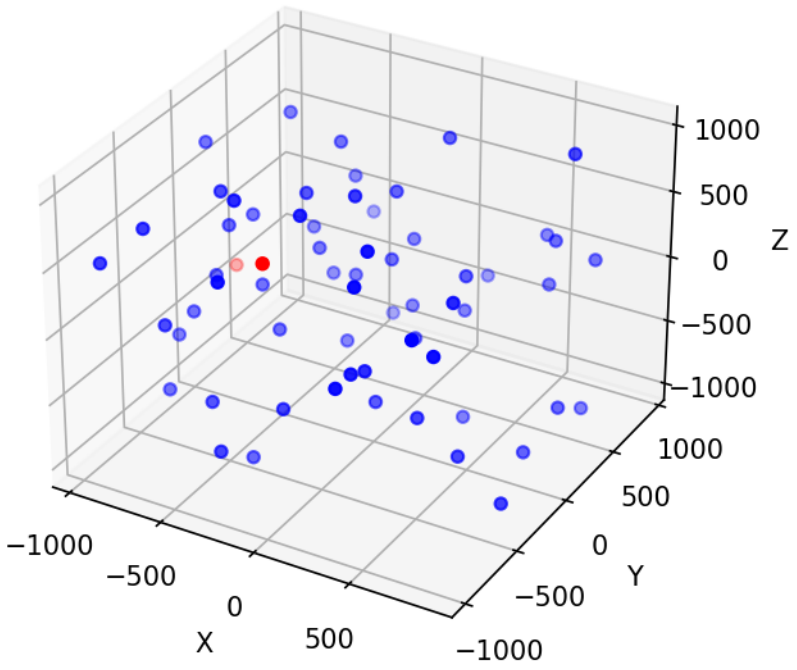


5.3. Test-case n = 64 (dimensi 3)

Bukti screenshoot

```
==================================================
================ NEAREST POINT ===================
==================================================
Please input the number of points: 64
Please input the dimension: 3
==================== DONE RANDOMIZING ====================
Do you want to see all the points that were generated?(y/n)
==========================================================
n
================ Divide and Conquer Solution ================
Your nearest points are :
[-466.52833013152554, -463.14575352843406, 368.2786551356569]
[-332.1250066519958, -455.11241900418327, 426.0129130802104]
Distance :  146.4992572210858
Euclidean distance computation count :  941
Execution time : 3.08  miliseconds
============================================================

==================== Brute Force Solution ====================
Your nearest points are :
[-466.52833013152554, -463.14575352843406, 368.2786551356569]
[-332.1250066519958, -455.11241900418327, 426.0129130802104]
Distance :  146.4992572210858
Euclidean distance computation count :  2017
Execution time : 10.16  miliseconds
============================================================
```

Hasil visualisasi:

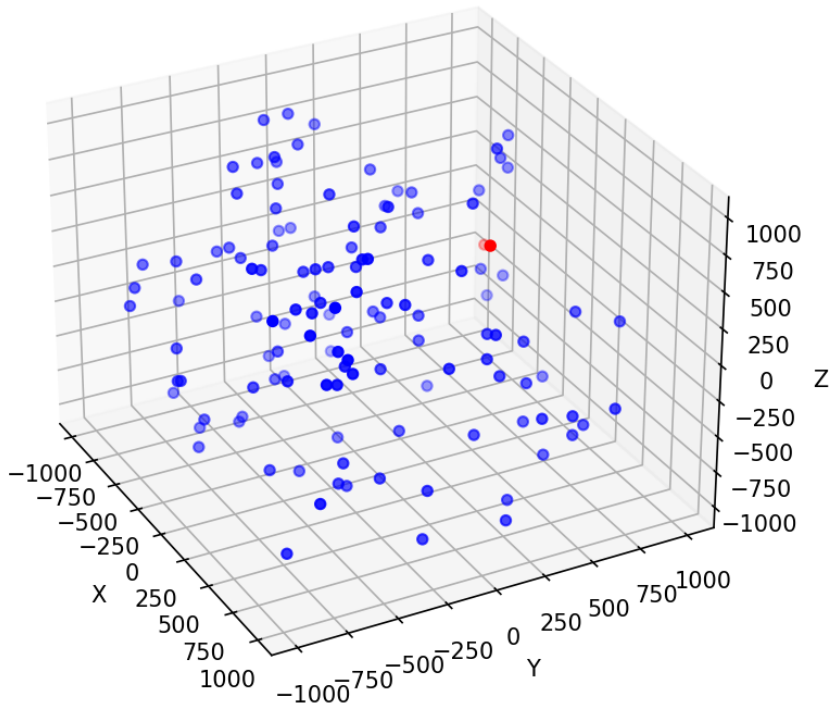


5.4. Test-case n = 128 (dimensi 3)

Bukti screenshoot

```
====================================================
=============== NEAREST POINT ======================
====================================================
Please input the number of points: 128
Please input the dimension: 3
=================== DONE RANDOMIZING ===============
Do you want to see all the points that were generated?(y/n)
====================================================
n
================ Divide and Conquer Solution ================
Your nearest points are :
[-647.6075375174395, 917.31987276376, -212.4423510709073]
[-619.0489841787856, 934.8116262650844, -208.26650985119932]
Distance :  33.74892678649268
Euclidean distance computation count :  1566
Execution time : 9.53  miliseconds
============================================================

=================== Brute Force Solution ===================
Your nearest points are :
[-647.6075375174395, 917.31987276376, -212.4423510709073]
[-619.0489841787856, 934.8116262650844, -208.26650985119932]
Distance :  33.74892678649268
Euclidean distance computation count :  8129
Execution time : 65.57  miliseconds
============================================================
```

Hasil visualisasi:

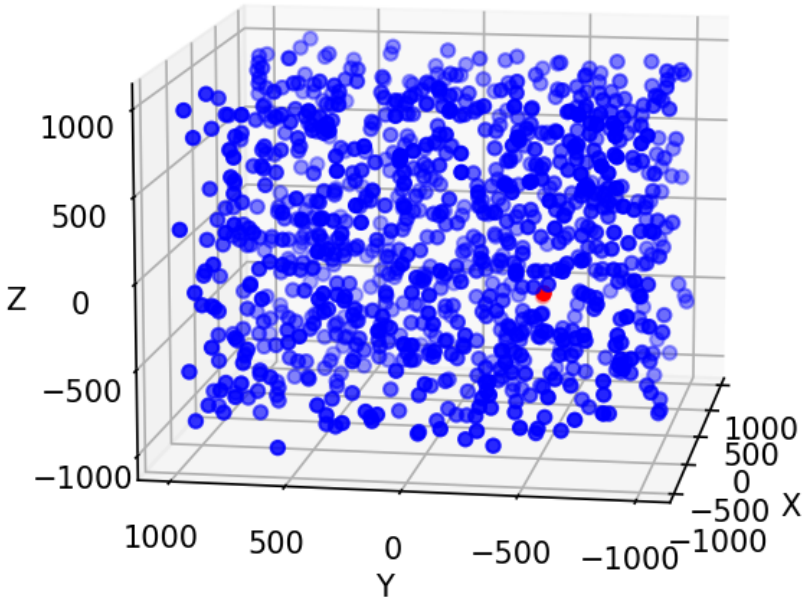


5.5. Test-case n = 1000 (dimensi 3)

Bukti screenshoot

```
======================================================
================ NEAREST POINT =======================
======================================================
Please input the number of points: 1000
Please input the dimension: 3
==================== DONE RANDOMIZING ====================
Do you want to see all the points that were generated?(y/n)
============================================================
n
================= Divide and Conquer Solution ================
Your nearest points are :
[-678.7045333478093, -538.2971112945401, -124.65121914536951]
[-675.7959891408498, -541.5876201790873, -106.89015641778781]
Distance :  18.29596751571815
Euclidean distance computation count :  42177
Execution time : 88.27  miliseconds
============================================================

==================== Brute Force Solution ====================
Your nearest points are :
[-678.7045333478093, -538.2971112945401, -124.65121914536951]
[-675.7959891408498, -541.5876201790873, -106.89015641778781]
Distance :  18.29596751571815
Euclidean distance computation count :  499501
Execution time : 2700.60  miliseconds
============================================================
```

Hasil visualisasi:



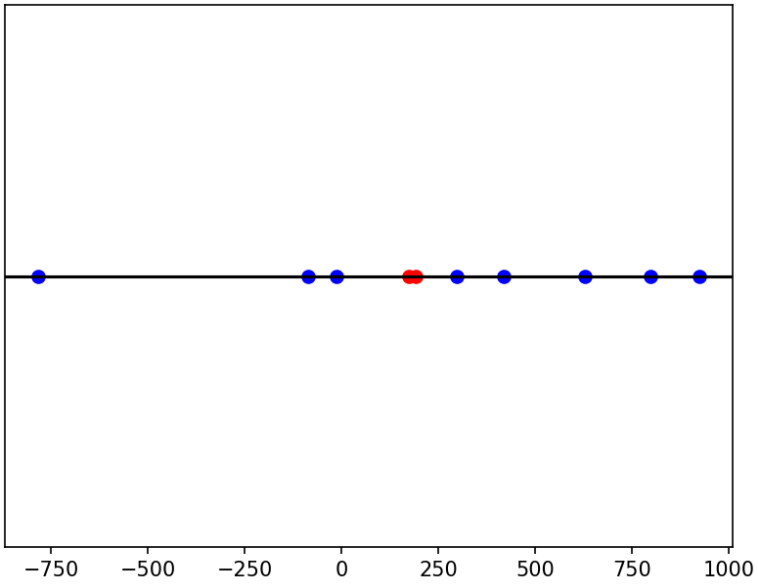
5.6. Test-case dimensi 1 (n = 10)

Bukti screenshot:

```
==========================================================
================ NEAREST POINT =======================
==========================================================
Please input the number of points: 10
Please input the dimension: 1
==================== DONE RANDOMIZING ====================
Do you want to see all the points that were generated?(y/n)
==========================================================
y
[419.8544783969064]
[-12.752474216278074]
[-86.90565328996217]
[173.39920139246374]
[296.2606898451338]
[191.00860135521657]
[798.0979234723598]
[627.5745310112634]
[-784.3918839921089]
[924.3354387855265]
================ Divide and Conquer Solution ================
Your nearest points are :
[173.39920139246374]
[191.00860135521657]
Distance :  17.609399962752832
Euclidean distance computation count :  7
Execution time : 0.07  miliseconds
==========================================================
```

```
==================== Brute Force Solution ==================
Your nearest points are :
[173.39920139246374]
[191.00860135521657]
Distance :  17.609399962752832
Euclidean distance computation count :  46
Execution time : 0.22  miliseconds
============================================================
```

Hasil visualisasi:

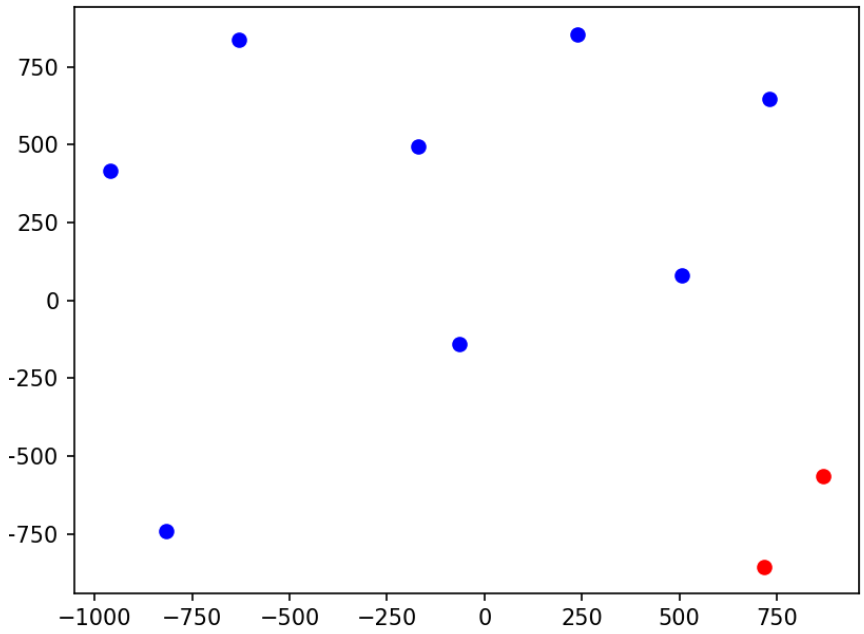


5.7. Test-case dimensi 2 (n = 10)

Bukti screenshot:

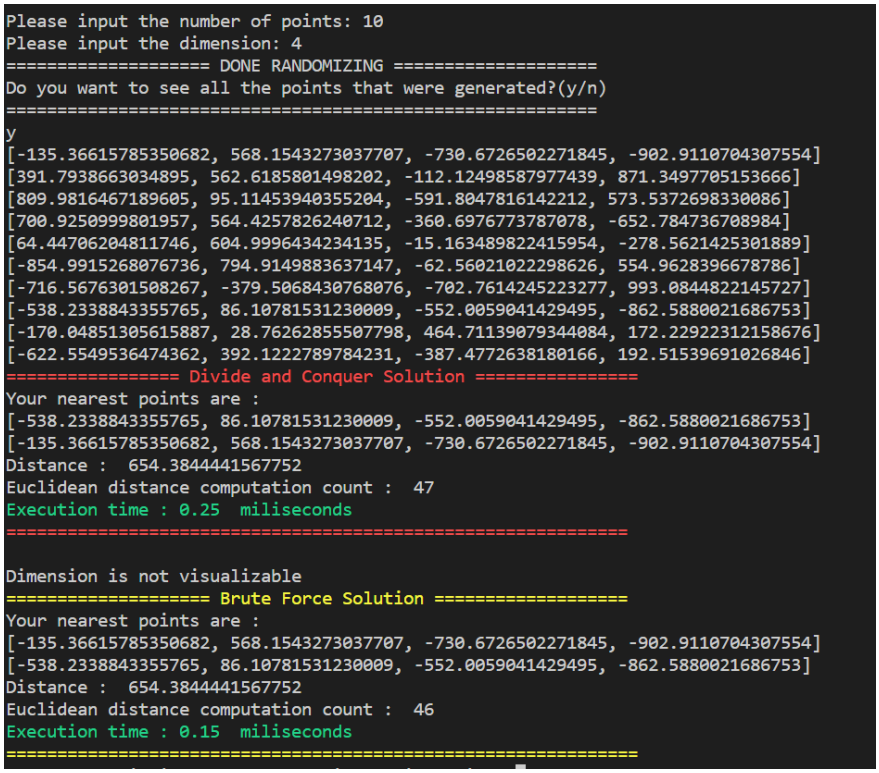
```
=========================================================
=============== NEAREST POINT ======================
=========================================================
Please input the number of points: 10
Please input the dimension: 2
=================== DONE RANDOMIZING ====================
Do you want to see all the points that were generated?(y/n)
=========================================================
n
================ Divide and Conquer Solution ================
Your nearest points are :
[716.2461395879018, -854.8727791725614]
[869.2879874627786, -564.1674060747182]
Distance :  328.52917853504755
Euclidean distance computation count :  25
Execution time : 0.17  miliseconds
=============================================================

=================== Brute Force Solution ====================
Your nearest points are :
[716.2461395879018, -854.8727791725614]
[869.2879874627786, -564.1674060747182]
Distance :  328.52917853504755
Euclidean distance computation count :  46
Execution time : 0.18  miliseconds
=============================================================
```

Hasil visualisasi:



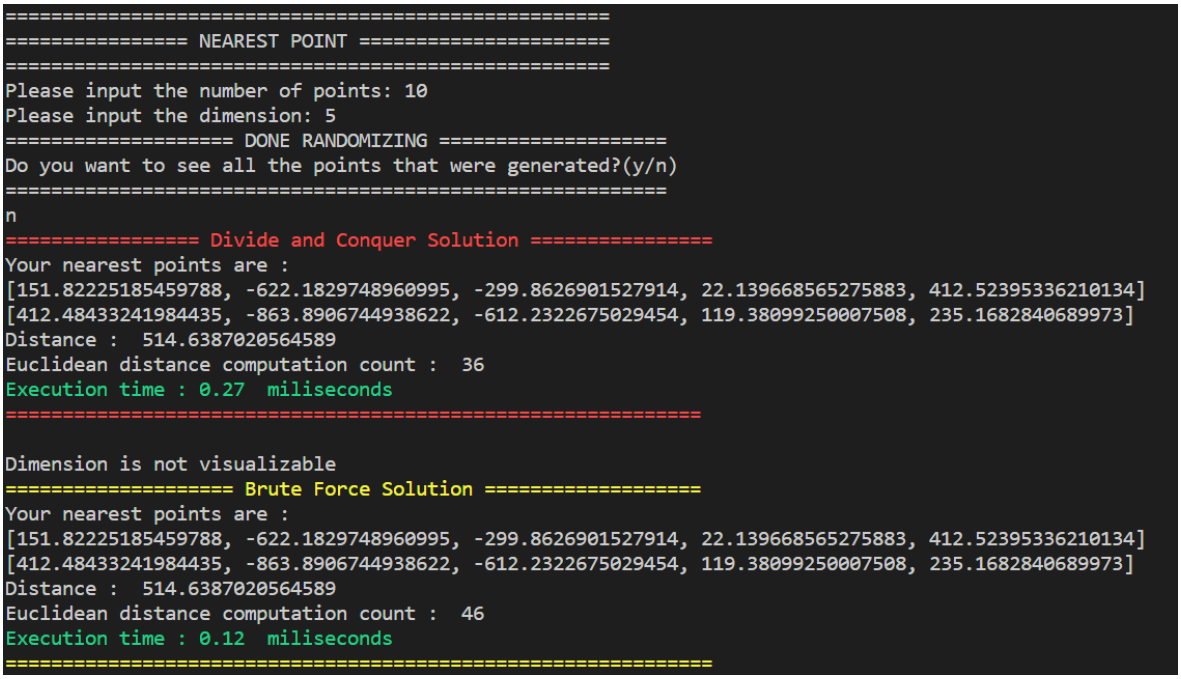
5.8. Test-case dimensi 4 (n = 10)

Bukti screenshot:



5.9. Test-case dimensi 5 (n = 10)

Bukti screenshot:



BAB 6
PENUTUP

6.1 Kesimpulan

Setelah melakukan berbagai eksplorasi terkait algoritma *divide and conquer* dan implementasinya pada persoalan pencarian pasangan titik terdekat, penulis menyimpulkan bahwa algoritma *divide and conquer* memiliki kompleksitas algoritma yang jauh lebih baik dibandingkan dengan algoritma *brute force*. Hal itu dapat dilihat dari *executable time* kedua algoritma tersebut.

Lampiran dan Daftar Pustaka

TABEL CHECKLIST SPEK

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	✔	

PEMBAGIAN TUGAS

NIM	**Nama**	**Tugas**
13521111	Tabitha Permalla	*Algoritma dan laporan dibagi rata*
13521139	Nathania Calista	

***Link repository* GitHub :**
https://github.com/nathaniacalista01/Tucil2_13521111_13521139

Referensi :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf
https://www.princewisnu.com/2020/06/pengertian-algoritma-merge-sort.html