LAPORAN TUGAS KECIL 2 IF2211 STRATEGI ALGORITMA

MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA DIVIDE AND CONQUER

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

Disusun oleh :

Vieri Fajar Firdaus 13521099

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2022/2023

Bab 1 Deskripsi Masalah

Mencari sepasang 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 sepasang 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 $P_1 = (x_1, y_1, z_1)$ dan $P_2 = (x_2, y_2, z_2)$ dihitung dengan rumus Euclidean berikut:

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

Buatlah program dalam dalam Bahasa C/C++/Java/Python/Golang/Ruby/Perl (pilih salah satu) untuk mencari sepasang titik yang jaraknya terdekat datu sama lain dengan menerapkan algoritma *divide and conquer* untuk penyelesaiannya, dan perbandingannya dengan Algoritma *Brute Force*.
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 Euclidian
- waktu riil dalam detik (spesifikasikan komputer yang digunakan)
- Bonus 1 (Nilai = 7,5) penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya.
- Bonus 2 (nilai = 7,5): Generalisasi program anda sehingga dapat mencari sepasang titik terdekat untuk sekumpulan vektor di R_n, setiap vektor dinyatakan dalam bentuk **x** = (x1, x2,…, xn)

Bab 2 Algoritma

Landasar teori yang akan digunakan pada penerapan strategi algoritma dalam mencari jarak minimum dari titik pada 3 dimesi yaitu: algoritma *Divide and Conquer* dan *Brute Force*

A. Definisi Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* merupakan algoritma umum yang digunakan untuk memecahkan suatu masalah besar menjadi beberapa masalah kecil dan menyelesaikan masalah kecil secara terpisah, kemudian menggabungkan solusi masalah kecil dan memecahkan masalah asli. Dari namanya Divide and Conquer dibagi menjadi tiga langkah dasar:

1. Divide: membagi masalah menjadi masalah-masalah yang lebih kecil. Biasanya menjadi setengahnya atau mendekati setengahnya
2. Conquer: ketika sebuah masalah sudah cukup kecil untuk diselesaikan, langsung selesaikan masalah tersebut.
3. Combine: ketika sebuah masalah sudah cukup kecil untuk diselesaikan, langsung selesaikan masalah tersebut.

Algoritma *Divide and Conquer* digunakan dalam banyak aplikasi, termasuk komputasi paralel, optimasi dan pembuatan keputusan. Beberapa contoh dari masalah yang dapat dipecahkan dengan penedakat ini meliputi *sorting,* pencarian elemen terntentu, penghitungan nilai fungsi matematika. Dalam implementasinya, algoritma *Divide and Conquer* dapat memberikan solusi yang efektif daripada pendekatan algoritma lainnya.

B. Implementasi *Divide and Conquer* pada algoritma *quicksort*

Terdapat algoritma pengurutan yang memiliki kompleksitas cukup cepat yaitu *quicksort*. *Quicksort* menggunakan prinsip *divide and conquer* dalam pengurutannya. Tahapan dari pengurutan *quicksort* adalah sebagai berikut :

1. Divide: pilih suatu elemen yang kita sebut *pivot,* kemudian kita bagi *array* menjadi dua sehingga *object* dalam array $\leq$ *pivot* dan yang lainnya selalu $>$ *pivot.*
2. Conquer: ketika *array* hanya memiliki satu elemen, *array* tersebut sudah terurut
3. Combine: gabunglah *subarray* dengan menempelkan hasil quicsort bagian kiri dan bagian kanan

Pada implementasi yang dilakukan sekarang ini, pivot secara awal dipilih dari indeks ke-0

C. Implementasi algoritma *closest pair*

Algoritma closest pair merupakan algortima pencarian pasangan titik dari himpunan semesta sehingga jarak pasangan titik tersebut minimum. Dengan menggunakan algoritma *brute force*, pencarian *closest pair* akan mudah dilakukan dengan cara melakukan pemilihan dua titik dari himpunan semesta, sehingga apabila terdapat n titik maka akan ada $C_2^n = \frac{n(n-1)}{2}$ kemungkinan, dengan mengecek $\frac{n(n-1)}{2}$ kemungkinan maka didapatkan *closest pair* dengan algoritma *brute force*. Namun algoritma ini dirasa cukup lama dibandingkan algoritma *Divide and Conquer*, untuk kompleksitas dari algoritma *brute force* adalah $O(n^2)$. Namun pada kali ini akan dicoba dengan menggunakan algoritma *divide and conquer,* untuk implementasinya pada pencarian *closest pair* sebagai berikut:

1. Divide: Sebelum melakukan algortima *DnC* pastika titik sudah terurut berdasarkan koordinat x, jika sudah terurut bagi kedua himpunan titik menjadi 2 bagian, dan cari closest pair dari kedua titik tersebut, serta pilih titik tengah sebagai median

2. Conquer: ketika jumlah titik ≤ 3 maka lakukanlah *brute force* untuk mendapatkan pasangan titik terpendek
3. Combine: cari closest pair dari himpunan titik satu dan himpunan titik dua (himpunan titik satu dan dua telah dilakukan pada bagian divide), pilih jarak terpendek dari kedua himpunan dan simpan dalam *temp_min*. Lakukan *traversal* sebanyak jumlah titik untuk mencari titik dengan selisih koordinat x terhadap titik median kurang dari *temp_min* lalu kumpulkan titik titik tersebut ke dalam array *temp_solution*. Setelah didapat array *temp_solution,* lakukan brute force pada *temp_solution* untuk mencari selisih yang lebih kecil dari *temp_min,* untuk mengoptimalisasi pencarian dilakukan pengecekan apabila selisih koordinat $x_2, x_3 \ldots x_n$ lebih dari temp_min maka tidak akan dilakukan penghitungan jarak

Untuk kompleksitas pada dimensi 3 yaitu $O(n) = n \log^2 n$ dan untuk kompleksitas pada dimensi-d yaitu $O(n) = n \log^{\mathrm{d}} n$

D. Algoritma Penggambaran titik pada 3D dan 2D

Untuk penggambaran titik pada 3D dan 2D digunakan library matplotlib untuk menggambarnya, untuk menandakan pasangan titik terdekat digambarkan dengan titik merah dan dihubungkan oleh garis berwarna merah.

Secara singkat kode akan mengelompokkkan koordinat x, koordinat y, dan koordinat z lalu akan dilakukan plot pada bidang 3D,untuk menandai pasangan titik terdekat akan digambarkan garis dengan warna merah setelah itu akan dilakukan pelabelan sumbu x, sumbu y, dan sumbu z. Untuk lebih lengkapnya kdoe dapat dilihat pada bagian *source code*

Bab 4 Kode Program dalam Bahasa Python

A. Kode main.py

```
from input import *
from point import *
from sorting import *
from plot import *
import time

def main() :

    point=[]
    (derajatcnt,titikcnt)=inp()
    point=pointinput(derajatcnt,titikcnt)
    sorting(point,0,len(point)-1,0)

    dncawal=time.time()
    iDNC,jDNC,solutionDNC,cntDNC=Closest(point,0,titikcnt-1)
    dncakhir=time.time()
    iBF,jBF,solutionBF,cntBF=brute_force(point,0,titikcnt-1)
    bfakhir=time.time()

    aa=(dncakhir-dncawal)
    bb=(bfakhir-dncakhir)
    print("Solusi koordinat titik berdasarkan DNF: ")
    print(point[iDNC])
    print(point[jDNC])
    print("Dengan jarak :",solutionDNC)
    print("Solusi koordinat titik berdasarkan BF: ")
    print(point[iBF])
    print(point[jBF])
    print("Dengan jarak :",solutionBF)
    print("DNC :",aa*1000,"ms")
    print("BF :",bb*1000,"ms")

    print("Banyak proses DNC:",cntDNC)
    print("Banyak proses BF:",cntBF)

    if(derajatcnt==2 or derajatcnt==3) :
        tampilkan=input("Apakah ingin menampilkan plot dalam 2D atau
3D? (y/n): ")
        if(tampilkan=="y") :
            if(derajatcnt==3) :
                plot3d(point,iDNC,jDNC)
            if(derajatcnt==2) :
                plot2d(point,iDNC,jDNC)

main()
```

B. Kode input.py

```
from function import *
from sorting import *
import random
def inp() :
    derajatcnt=checkkint("Masukkan banyak derajat : ")
    titikcnt=checkkint("Masukkan banyak titik : ")
    return (derajatcnt,titikcnt)

def pointinput(derajatcnt,titikcnt) :
    arr=[]
    for i in range(titikcnt) :
        arr.append([])
        for j in range(derajatcnt) :
            arr[i].append(random.uniform(-100,100))
    return arr
```

C. Kode sorting.py

```
def swap(a,b) :
    return (b,a)

def partition(arr,l,r,param) :
    pivot=arr[r][param]

    i=l-1

    for j in range(l,r) :
        if arr[j][param]<pivot :
            i+=1
            arr[i],arr[j]=swap(arr[i],arr[j])

    arr[i+1],arr[r]=swap(arr[i+1],arr[r])
    return i+1

def sorting(arr,l,r,param) :
    if l<r :
        pi=partition(arr,l,r,param)
        sorting(arr,l,pi-1,param)
        sorting(arr,pi+1,r,param)
```

D. Kode point.py

```
from function import *
from brute_force import *

def Closest(point,start,end) :
```

```
    point_i=-1
    point_j=-1
    solution=float('inf')
    cnt_distance=0
    if(end-start<=3) :
        return brute_force(point,start,end)

    mid=(end+start)//2
    min_left=Closest(point,start,mid)
    min_right=Closest(point,mid+1,end)
    cnt_distance=min_left[3]+min_right[3]

    if(min_left[2]<min_right[2]) :
        point_i=min_left[0]
        point_j=min_left[1]
        solution=min_left[2]
    else :
        point_i=min_right[0]
        point_j=min_right[1]
        solution=min_right[2]

    temp_solution=[mid]

    for i in range(start,end+1) :
        if(abs(point[i][0]-point[mid][0])<solution and i!=mid) :
        # if(distance(point[i],point[mid])<solution and i!=mid) :
            temp_solution.append(i)


    for i in range(len(temp_solution)) :
        for j in range(i+1,len(temp_solution)) :

if(deltacondition(point,temp_solution[i],temp_solution[j],solution)) :
                cnt_distance+=1

if(distance(point[temp_solution[i]],point[temp_solution[j]])<solution)
:

solution=distance(point[temp_solution[i]],point[temp_solution[j]])
                    point_i=temp_solution[i]
                    point_j=temp_solution[j]

    return (point_i,point_j,solution,cnt_distance)
```

E. Kode brute_force.py

```
from function import *

def brute_force(point,start,end) :
```

```
    cnt_distance=0
    solution=float('inf')
    point_i=start
    point_j=end

    for i in range(start,end+1) :
        for j in range(i+1,end+1) :
            cnt_distance+=1
            if(distance(point[i],point[j])<solution) :
                solution=distance(point[i],point[j])
                point_i=i
                point_j=j

    return (point_i,point_j,solution,cnt_distance)
```

F. Kode plot.py

```
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D
import numpy as np

def plot3d(arr,i,j) :
    xs=[]
    ys=[]
    zs=[]
    for ii in range(len(arr)) :
        xs.append(arr[ii][0])
        ys.append(arr[ii][1])
        zs.append(arr[ii][2])
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(xs,ys,zs)


ax.scatter([arr[i][0],arr[j][0]],[arr[i][1],arr[j][1]],[arr[i][2],arr[j
][2]],c='red',s=30)

ax.plot([arr[i][0],arr[j][0]],[arr[i][1],arr[j][1]],[arr[i][2],arr[j][2
]],color='red')
    ax.set_xlabel("Sumbu X")
    ax.set_ylabel("Sumbu Y")
    ax.set_zlabel("Sumbu Z")
    plt.show()

def plot2d(arr,i,j) :
    xs=[]
    ys=[]
    for ii in range(len(arr)) :
```

```
        xs.append(arr[ii][0])
        ys.append(arr[ii][1])
    plt.scatter(xs,ys)

plt.scatter([arr[i][0],arr[j][0]],[arr[i][1],arr[j][1]],c='red',s=30)
    plt.plot([arr[i][0],arr[j][0]],[arr[i][1],arr[j][1]],color='red')
    plt.show()
```

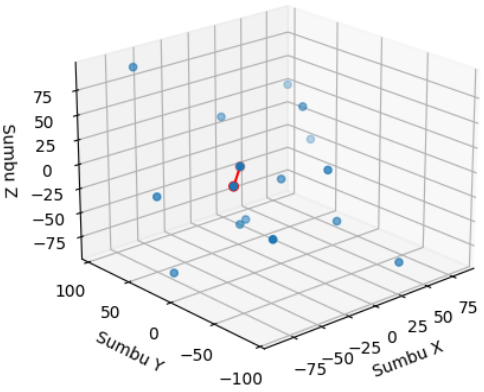
G. Kode function.py

```
import random
def swap(a,b) :
    return (b,a)

def distance(a,b) :
    res=0.0
    for i in range(len(a)) :
        res+=(a[i]-b[i])**2
    return res**0.5

def checkkint(msg):
    while 1:
        try:
            n = input(msg)
            return(int(n))
        except ValueError:
            print("Masukkan sebuah bilangan bulat")

def deltacondition(point,i,mid,temp_min) :
    res=True
    for j in range(len(point[0])) :
        if(abs(point[i][j]-point[mid][j])<temp_min) :
            res=True
        else :
            return False
    return res
```

Bab 5 Dokumentasi

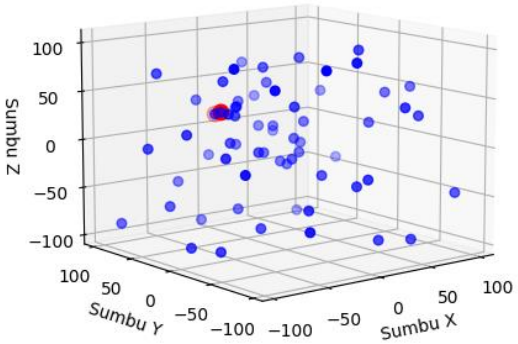
A. Banyak titik 16 dengan banyak derajat tiga

```
Masukkan banyak derajat : 3
Masukkan banyak titik : 16
Solusi koordinat titik berdasarkan DNF:
[-83.74523257820545, -50.6509896247497, 30.920949304432668]
[-76.18717065092028, -48.319336299267675, 46.5942854993302]
Dengan jarak : 17.556035281497753
Solusi koordinat titik berdasarkan BF:
[-83.74523257820545, -50.6509896247497, 30.920949304432668]
[-76.18717065092028, -48.319336299267675, 46.5942854993302]
Dengan jarak : 17.556035281497753
DNC : 0.0 ms
BF : 0.8187294006347656 ms
Banyak proses DNC: 28
Banyak proses BF: 120
Apakah ingin menampilkan plot dalam 2D atau 3D? (y/n): y
```



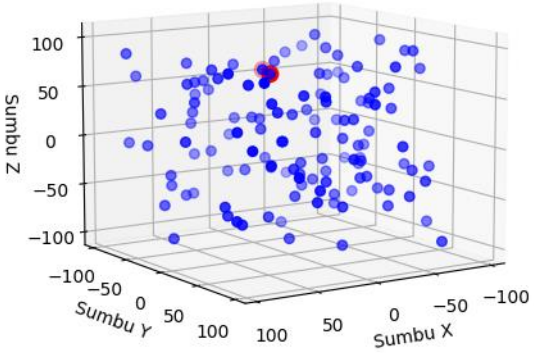
B. Banyak titik 64 dengan banyak derajat tiga

```
Masukkan banyak derajat : 3
Masukkan banyak titik : 64
Solusi koordinat titik berdasarkan DNF:
[-37.524671796563865, 40.25203336918989, 29.121508734949145]
[-35.67160810140339, 35.08197912777146, 31.593691966945585]
Dengan jarak : 6.022872724049115
Solusi koordinat titik berdasarkan BF:
[-37.524671796563865, 40.25203336918989, 29.121508734949145]
[-35.67160810140339, 35.08197912777146, 31.593691966945585]
Dengan jarak : 6.022872724049115
DNC : 0.0 ms
BF : 0.0 ms
Banyak proses DNC: 111
Banyak proses BF: 2016
```



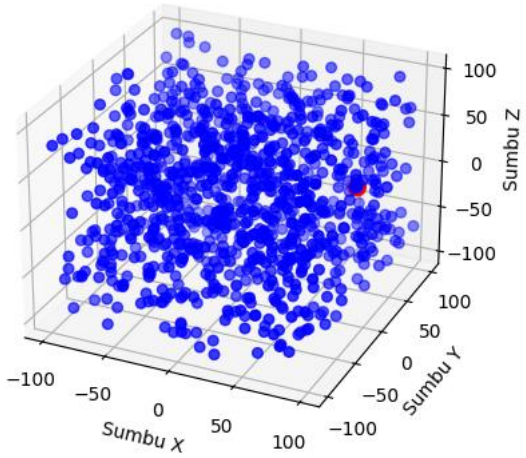
C. Banyak titik 128 dengan banyak derajat tiga

```
Masukkan banyak derajat : 3
Masukkan banyak titik : 128
Solusi koordinat titik berdasarkan DNF:
[37.37455266373718, 32.507045961228215, 78.69274333541048]
[37.992499509048855, 23.28123439543768, 81.49334581717662]
Dengan jarak : 9.661305895787573
Solusi koordinat titik berdasarkan BF:
[37.37455266373718, 32.507045961228215, 78.69274333541048]
[37.992499509048855, 23.28123439543768, 81.49334581717662]
Dengan jarak : 9.661305895787573
DNC : 9.558439254760742 ms
BF : 10.388851165771484 ms
Banyak proses DNC: 241
Banyak proses BF: 8128
```



D. Banyak titik 1000 dengan banyak derajat tiga

```
Masukkan banyak derajat : 3
Masukkan banyak titik : 1000
Solusi koordinat titik berdasarkan DNF:
[97.76904142763146, -14.49532415923332, 45.488664128070326]
[98.94925956831466, -14.519653252540593, 44.40367306463017]
Dengan jarak : 1.6033441215546504
Solusi koordinat titik berdasarkan BF:
[97.76904142763146, -14.49532415923332, 45.488664128070326]
[98.94925956831466, -14.519653252540593, 44.40367306463017]
Dengan jarak : 1.6033441215546504
DNC : 58.13169479370117 ms
BF : 669.3816184997559 ms
Banyak proses DNC: 1898
Banyak proses BF: 499500
```



E. Banyak titik 16 dengan banyak derajat empat

```
Masukkan banyak derajat : 4
Masukkan banyak titik : 16
Solusi koordinat titik berdasarkan DNF:
[-5.562150239023623, 80.97852567007305, 65.61773732546698, -58.93028005019359]
[14.781027415861473, 58.270825735206216, 90.63330186567092, -96.4682421265024]
Dengan jarak : 54.44595099305067
Solusi koordinat titik berdasarkan BF:
[-5.562150239023623, 80.97852567007305, 65.61773732546698, -58.93028005019359]
[14.781027415861473, 58.270825735206216, 90.63330186567092, -96.4682421265024]
Dengan jarak : 54.44595099305067
DNC : 4.065275192260742 ms
BF : 0.0 ms
Banyak proses DNC: 31
Banyak proses BF: 120
```

F. Banyak titik 100 dengan banyak derajat 10

```
Masukkan banyak derajat : 10
Masukkan banyak titik : 100
Solusi koordinat titik berdasarkan DNF:
[-22.731144715160895, -76.62314291663569, -53.39423772901846, 24.28337325418886, -33.803956393222705, 65.12427901649497, 54.22235351072112, -91.803854723453
81, 4.055186175656871, 71.0820522080025]
[-16.73280066679213, -99.95440182414103, -57.369363493689775, 13.523559352263575, -82.39969048308318, 58.972398750798845, 81.4495230510253, -97.028336161421
36, 37.016111526740644, 95.74355119421193]
Dengan jarak : 74.66270887019313
Solusi koordinat titik berdasarkan BF:
[-22.731144715160895, -76.62314291663569, -53.39423772901846, 24.28337325418886, -33.803956393222705, 65.12427901649497, 54.22235351072112, -91.803854723453
81, 4.055186175656871, 71.0820522080025]
[-16.73280066679213, -99.95440182414103, -57.369363493689775, 13.523559352263575, -82.39969048308318, 58.972398750798845, 81.4495230510253, -97.028336161421
36, 37.016111526740644, 95.74355119421193]
Dengan jarak : 74.66270887019313
DNC : 11.083841323852539 ms
BF : 13.93437385559082 ms
Banyak proses DNC: 787
Banyak proses BF: 4950
```

G. Banyak titik 200 dengan banyak derajat 20

```
Masukkan banyak derajat : 20
Masukkan banyak titik : 200
Solusi koordinat titik berdasarkan DNF:
[-16.899379505721797, -47.14185949376939, 98.14139738150331, -42.82470116530761, 43.026584023795266, 24.322697935275883, 32.5447861179054, -23.1547387468378
64, 56.44339754855346, -60.32127555075797, 1.4999658142055665, 26.66171091634763, 3.5775286852803845, -75.47822409197468, 40.799230066826766, 32.19726335235
745, 20.091453457509175, -1.2048498850777207, 80.65013506245978, 59.26986984942505]
[12.514119553979654, -67.66097822039168, 85.68071080464492, -42.87312678662782, -16.38907704274999, -2.2734770218656593, 67.09994122038293, 17.4420056777402
4, 80.30962433807161, -63.50928637353142, 5.805147771184167, 37.403242556756595, 20.083276507595244, -64.07626779546388, 23.079863966238605, 28.640830062416
995, 25.239084552964712, 79.76227592164585, 71.4100414571673, -44.666728868082785]
Dengan jarak : 165.6330405712123
Solusi koordinat titik berdasarkan BF:
[-16.899379505721797, -47.14185949376939, 98.14139738150331, -42.82470116530761, 43.026584023795266, 24.322697935275883, 32.5447861179054, -23.1547387468378
64, 56.44339754855346, -60.32127555075797, 1.4999658142055665, 26.66171091634763, 3.5775286852803845, -75.47822409197468, 40.799230066826766, 32.19726335235
745, 20.091453457509175, -1.2048498850777207, 80.65013506245978, 59.26986984942505]
[12.514119553979654, -67.66097822039168, 85.68071080464492, -42.87312678662782, -16.38907704274999, -2.2734770218656593, 67.09994122038293, 17.4420056777402
4, 80.30962433807161, -63.50928637353142, 5.805147771184167, 37.403242556756595, 20.083276507595244, -64.07626779546388, 23.079863966238605, 28.640830062416
995, 25.239084552964712, 79.76227592164585, 71.4100414571673, -44.666728868082785]
Dengan jarak : 165.6330405712123
DNC : 301.77807807922363 ms
BF : 90.28792381286621 ms
Banyak proses DNC: 29896
Banyak proses BF: 19900
```

H. Banyak titik 1000 dengan derajat empat

```
Masukkan banyak derajat : 4
Masukkan banyak titik : 1000
Solusi koordinat titik berdasarkan DNF:
[-18.63958106387173, -70.55386925285099, 78.58576535955368, -37.13859655649428]
[-18.348240590945323, -71.0304567126193, 78.05533180933838, -41.17063102737397]
Dengan jarak : 4.104957564156965
Solusi koordinat titik berdasarkan BF:
[-18.63958106387173, -70.55386925285099, 78.58576535955368, -37.13859655649428]
[-18.348240590945323, -71.0304567126193, 78.05533180933838, -41.17063102737397]
Dengan jarak : 4.104957564156965
DNC : 106.2312126159668 ms
BF : 823.1801986694336 ms
Banyak proses DNC: 2015
Banyak proses BF: 499500
```

Bab 6 Tabel Penilaian

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa ada kesalahan	V	
2	Program berhasil running	V	
3	Program dapat menerima masukan dan menuliskan luaran	V	
4	Luaran program sudah benar (solusi *closest pair* sudah benar)	V	
5	Bonus 1 dikerjakan	V	
6	Bonus 2 dikerjakan	V	

Bab 7 Kesimpulan

Dalam melakukan pencari *closest pair* terdapat berbagai macam metode yang dapat diaplikasikan, diantaranya *brute force* dan *divide and conquer*. Salah satunya adalah dengan menggunakan metode *divide and conquer:* dengan metode divide, conquer dan combine. Namun untuk beberapa kasus penggunaan brute force dirasa lebih cepat dibandingkan dnc untuk kasus dengan derajat tinggi (lebih dari 20)

Pada Tugas Kecil 2 Strategi Algoritma ini, penulis membuat aplikasi pencarian *closest pair*. Aplikasi dibuat dengan menggunakan bahasa python, bahasa ini dipilih karena terdapat library matplotlib untuk menggambarkan titik pada bangun 3 dimensi. Program ini dirasa cukup efektif namun masih belum begitu baik, penulis berharap bisa meningkatkan efektivitas program

Bab 8 Referensi

- http://www.niser.ac.in/~aritra/CG/ClosestPairProblem.pdf
- http://people.csail.mit.edu/indyk/6.838-old/handouts/lec17.pdf
- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf

Link Repository Github : https://github.com/vierifirdaus/Tucil2_13521099