MENYELESAIKAN PERMASALAHAN *CLOSEST-PAIR* DENGAN ALGORITME *DIVIDE AND CONQUER* DALAM BAHASA PEMROGRAMAN **PYTHON**

Disusun untuk memenuhi laporan tugas mata kuliah IF2211
Strategi Algoritma semester 4 di Institut Teknologi Bandung.

Disusun oleh:

Ryan Samuel Chandra
13521140

TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

Jl. Ganesa No. 10, Lb. Siliwangi, Kecamatan Coblong,
Kota Bandung, Jawa Barat, 40132

2023

PRAKATA

Syukur yang begitu luar biasa saya panjatkan kepada Tuhan Yang Maha Esa, sebab berkat-Nya telah membawa saya sampai ke titik ini. Pada akhirnya, dengan bangga, saya menyelesaikan laporan ini tepat waktu, setelah seminggu penuh melakukan eksplorasi mandiri dan pemrograman secara rutin. Tidak lupa, saya mengucapkan terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc., serta para asisten mata kuliah IF2211 Strategi Algoritma, yang telah membagikan ilmu serta memberikan kesempatan bagi saya untuk belajar melalui tugas ini. Terakhir, saya menyampaikan rasa sayang pada orang tua dan teman-teman tercinta. Tanpa dukungan mereka, laporan ini tidak akan terselesaikan.

Berikut disajikan laporan lengkap tentang "Menyelesaikan Permasalahan *Closest-Pair* dengan Algoritme *Divide and Conquer* dalam Bahasa Pemrograman Python". Laporan ini selain dibuat agar dapat bermanfaat bagi masyarakat, juga secara khusus disusun untuk memenuhi salah satu tugas mata kuliah IF2211 Strategi Algoritma di semester 4 Teknik Informatika Institut Teknologi Bandung.

Dengan referensi yang terkumpul dari berbagai sumber, dilanjutkan beberapa kali melewati gagal-berhasilnya percobaan, saya berusaha sebaik mungkin untuk memenuhi tujuan pembuatan yang telah ditetapkan meskipun harus bekerja di tengah hambatan dari lingkungan dan jadwal yang padat.

Saya berharap laporan ini dapat menjadi sesuatu yang memuaskan bagi semua pembaca. Namun, saya pun menyadari bahwa laporan ini masih jauh dari kesempurnaan. Untuk itu, saya sangat berterima kasih apabila terdapat kritik dan saran yang bersifat konstruktif untuk perkembangan percobaan saya. Mohon maaf apabila ada kesalahan kata. Sekian dari saya, selamat membaca.

Bandung, 28 Februari 2023

Penyusun Laporan

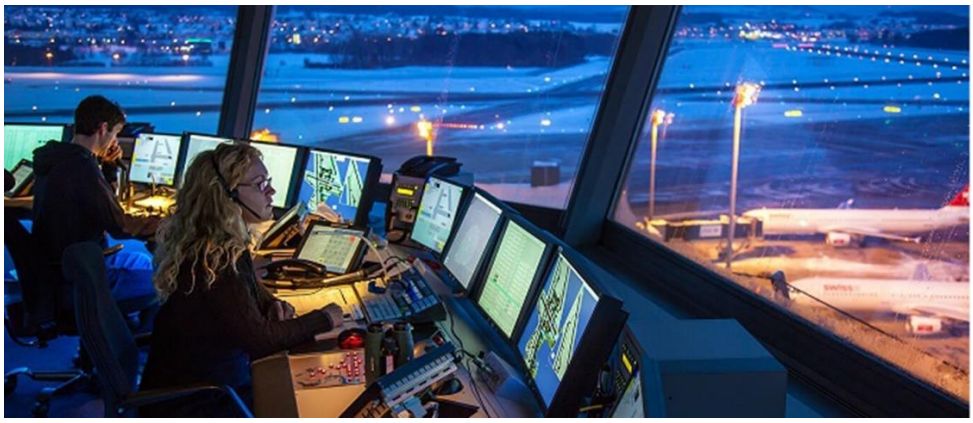
Daftar Isi

BAB 1
TINJAUAN PUSTAKA

1.1 *Closest-Pair Problem* (Permasalahan Pasangan Titik-Terdekat)

Mencari pasangan titik yang jaraknya terdekat, adalah salah satu persoalan terdahulu pada geometri komputasional. Masalahnya adalah: diberikan himpunan Q yang berisi $n \geq 2$ buah titik berdimensi tertentu, untuk kemudian dicari pasangan mana yang memiliki jarak Euclidean terdekat [7].

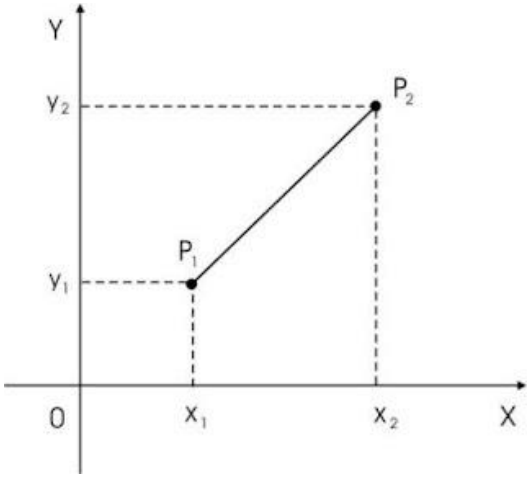
Permasalahan ini muncul di sejumlah kondisi dalam kehidupan nyata. Misalnya, dalam kontrol lalu lintas udara, seorang pemandu pasti akan memantau jarak pesawat yang mendekat ke pesawat lainnya, karena hal ini mengindikasikan kemungkinan tabrakan [8].



Gambar 1.1 Ruang Kontrol Lalu Lintas Udara
(Sumber: https://flylevel.aero/courses/air-traffic-controller/)

1.2 Menghitung Jarak Dua Titik pada Bidang Koordinat Cartesius

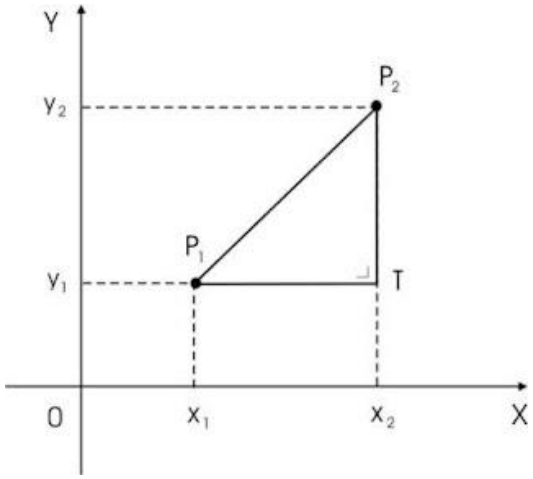
Sistem koordinat Cartesius mulai populer ketika Rene Descartes bersama rekannya Piere de Fermet memperkenalkannya sekitar pertengahan abad ke-17 [5]. Sistem koordinat ini terdiri dari dua buah garis (sumbu), yaitu garis mendatar (sumbu X), dan garis tegak (sumbu Y). Letak sebuah titik pada koordinat Cartesius diwakili oleh pasangan terurut (x, y), dengan x disebut absis dan y disebut ordinat. Contoh: $P_1(x_1, y_1)$ dan $P_2(x_2, y_2)$.



(Sumber: https://www.tipsbelajarmatematika.com/2017/05/cara-menentukan-jarak-dua-titik-pada.html)

Gambar 1.2 Ilustrasi Dua Titik pada Bidang Koordinat Cartesius

Jarak dua titik adalah garis hubung terpendek antara kedua titik tersebut. Misalkan pada gambar di atas, dinamai garis P_1P_2. Jika dua buah garis imajiner ditambahkan sehingga membentuk pola segitiga seperti gambar di bawah ini, dapat dihitung panjang garis P_1P_2 dengan memanfaatkan teorema Pythagoras [5].



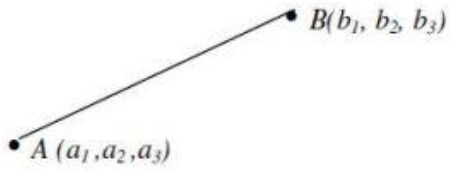
Gambar 1.3 Ilustrasi Garis Bantu untuk Menghitung Jarak Dua Titik
(Sumber: https://www.tipsbelajarmatematika.com/2017/05/cara-menentukan-jarak-dua-titik-pada.html)

$$(P_1P_2)^2 = (P_1T)^2 + (P_2T)^2$$

$$(P_1P_2) = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

atau sama dengan $$\mathbf{(P_1P_2) = (x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Jika dua titik terletak pada bidang koordinat Cartesius tiga dimensi, misalnya $A(a_1, a_2, a_3)$ dan $B(b_1, b_2, b_3)$, maka rumus jarak Euclidean sebelumnya dimodifikasi menjadi:



$$\text{Panjang } AB = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2}$$

Gambar 1.4 Rumus Menghitung Jarak Dua Titik pada Bidang Tiga Dimensi
(Sumber: https://renoyudistira1412.wordpress.com/jarak-titik-garis-dan-bidang/)

1.3 Algoritme *Brute Force*

Samuel R. dalam tulisannya, "Implementasi Algoritme *Brute Force* dalam Bahasa Pemrograman C untuk Menyelesaikan Permainan Kartu 24", berusaha mendeskripsikan algoritme *brute force* dengan lengkap. Berikut kutipan yang diambil dari karya tersebut:

Algoritme *brute force* adalah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu persoalan [1] [4]. Biasanya algoritme ini didasarkan pada pernyataan pada persoalan (*problem statement*) dan definisi/konsep yang dilibatkan. *Brute force* memecahkan persoalan dengan sangat sederhana, langsung, dan jelas caranya.

Algoritme *brute force* umumnya tidak "cerdas" dan tidak mangkus, karena ia membutuhkan volume komputasi yang besar dan waktu yang lama dalam penyelesaiannya. Kata "*force*" mengindikasikan tenaga ketimbang otak. Maka dari itu, kadang-kadang algoritme ini disebut juga algoritme naif [1] [4].

Algoritme *brute force* lebih cocok untuk persoalan yang masukannya kecil, serta sering digunakan sebagai basis pembanding dengan algoritme lain yang lebih mangkus. Meskipun bukan metode *problem solving* yang mangkus, hampir semua persoalan dapat diselesaikan secara akurat dengan algoritme ini [1] [4].

1.4 Algoritme *Divide and Conquer*

Sesuai namanya, algoritme *divide and conquer* memiliki dua bagian utama. Pertama, *divide*, berarti membagi persoalan (*problem*) menjadi beberapa upapersoalan (*subproblem*) yang memiliki kemiripan karakteristik dengan persoalan semula, namun ukurannya lebih kecil. Sedangkan *conquer* (*solve*) berarti menyelesaikan masing-masing upapersoalan secara langsung jika sudah berukuran kecil [3]. Setelah kedua tahap tersebut dilakukan, terdapat tambahan langkah terakhir, *combine*, yaitu menggabungkan solusi masing-masing upapersoalan sehingga membentuk solusi persoalan semula [3].

Adapun objek persoalan yang dibagi, biasanya merupakan masukan (input) atau *instances* persoalan yang berukuran *n*, seperti tabel (larik), matriks, eksponen, polinom, dan lain-lain. Kemiripan karakteristik antara persoalan dengan upapersoalannya membuat metode *divide and conquer* lebih natural diungkapkan dalam skema rekursif [3].

Dikutip dari sumber [3], skema umum algoritme *divide and conquer* adalah sebagai berikut:

```
procedure DIVIDEandCONQUER (input P : problem, n : integer)
    {Menyelesaikan persoalan P berukuran n dengan algoritme divide and conquer}
{KAMUS LOKAL}
    r, i : integer
{ALGORITME}
    if (n ⩽ n0) then  {ukuran persoalan P sudah cukup kecil}
        SOLVE persoalan P yang berukuran n ini
    else
        DIVIDE menjadi r upapersoalan P1,P2,…,Pr yang berukuran n1,n2,…,nr
        i traversal [1..r]
            DIVIDEandCONQUER(Pi, ni)

        COMBINE solusi dari P1,P2,…,Pr menjadi solusi persoalan semula
```

Kompleksitas algoritme *divide and conquer*[3] adalah sebagai berikut :

$$T(n) = \begin{cases} g(n), & n \le n_0 \\ T(n_1) + T(n_2) + \cdots + T(n_3) + f(n), & n > n_0 \end{cases}$$

dengan T(n) adalah kompleksitas waktu penyelesaian persoalan semula, g(n) adalah kompleksitas waktu untuk "SOLVE" jika *n* sudah berukuran kecil, T(n_i) adalah kompleksitas waktu menyelesaikan upapersoalan *i*, dan f(n) adalah kompleksitas waktu untuk "COMBINE".

Kasus khusus adalah persoalan selalu dibagi menjadi dua upapersoalan yang berukuran sama [3]. Maka kompleksitasnya menjadi:

$$T(n) = \begin{cases} g(n), & n \leq n_0 \\ 2T\left(\frac{n}{2}\right) + f(n), & n > n_0 \end{cases}$$

Beberapa persoalan yang dapat diselesaikan menggunakan *divide and conquer* [3]:

1. Mencari nilai minimum dan nilai maksimum.
2. Menghitung perpangkatan.
3. Persoalan pengurutan (*sorting*): *mergesort* dan *quicksort*.
4. Mencari sepasang titik terdekat (*closest-pair problem*).
5. *Convex hull*.
6. Perkalian matriks.
7. Perkalian bilangan bulat besar.
8. Perkalian dua buah polinom.

BAB 2
IMPLEMENTASI

2.1 Algoritme *Brute Force* dalam Menyelesaikan *Closest-Pair Problem*

Seperti pada umumnya, solusi menggunakan *brute force* tentu lebih mudah untuk dikomputasikan. Tanpa banyak berpikir, komputer diminta untuk membangkitkan *n* buah koordinat titik tiga dimensi. Setiap titik harus memiliki nilai pada masing-masing sumbunya, yaitu *x*, *y*, dan *z*. Maka dari itu, disediakan tempat bagi komputer, berupa senarai dua dimensi berukuran $n \times 3$, untuk menyimpan bilangan pecahan acak (*random*) yang dibangkitkan.

Awalnya, sebuah kalang (*loop*) difungsikan untuk "mengambil" titik pertama pada matriks. Kalang berikutnya akan mulai dari elemen titik kedua, bertugas mengiterasi satu per satu elemen sampai akhir, sekaligus memanggil fungsi untuk menghitung jarak titik-titik tersebut terhadap titik yang dipegang kalang pertama.

Kemudian, proses akan diulangi dengan kalang pertama memegang elemen kedua, kalang kedua mulai dari elemen ketiga dan kembali mengiterasi sampai akhir; demikian seterusnya sampai kalang pertama keluar dari larik. Bisa dipastikan, suatu titik tidak akan dihitung jaraknya dengan dirinya sendiri (sebab menghasilkan 0), juga dengan titik yang sudah pernah menjadi pasangannya. Ini adalah upaya mengefisienkan proses *brute force*, meskipun sudah tentu kalah cepat dengan *divide and conquer*. Berikut adalah ilustrasinya:

```
Titik:
[-42, 44, 23]
[11, -16, 48]
[-7, 31, 11]
[23, 11, -15]
[-50, 20, -9]
Hitung: jarak antara [-42, 44, 23] dan [11, -16, 48] = 83.86894538504703
Jarak minimum saat ini: 83.86894538504703
Hitung: jarak antara [-42, 44, 23] dan [-7, 31, 11] = 39.21734310225516
Jarak minimum saat ini: 39.21734310225516
Hitung: jarak antara [-42, 44, 23] dan [23, 11, -15] = 82.20705565825844
Hitung: jarak antara [-42, 44, 23] dan [-50, 20, -9] = 40.792156108742276
Hitung: jarak antara [11, -16, 48] dan [-7, 31, 11] = 62.465990746965666
Hitung: jarak antara [11, -16, 48] dan [23, 11, -15] = 69.58448102845921
Hitung: jarak antara [11, -16, 48] dan [-50, 20, -9] = 90.91754506144565
Hitung: jarak antara [-7, 31, 11] dan [23, 11, -15] = 44.45222154178574
Hitung: jarak antara [-7, 31, 11] dan [-50, 20, -9] = 48.68264577855234
Hitung: jarak antara [23, 11, -15] dan [-50, 20, -9] = 73.79701890998037

Jarak dua titik terdekat: 39.21734 satuan,
yaitu antara titik:
 [-42, 44, 23] dan [-7, 31, 11]
```

Gambar 2.1 Ilustrasi Algoritme *Brute Force* dalam *Closest-Pair*
(Sumber: Dokumentasi penulis)

Secara matematis, proses seperti ini secara eksak membutuhkan sebanyak $\frac{n \times (n-1)}{2}$ buah perhitungan jarak antartitik, menghasilkan kompleksitas algoritme: $O(n^2)$. Jika masukannya kecil, misalnya $n = 8$, maka dibutuhkan hanya 28 perhitungan. Namun, jika $n = 2048$, dibutuhkan 2.096.128 perhitungan!

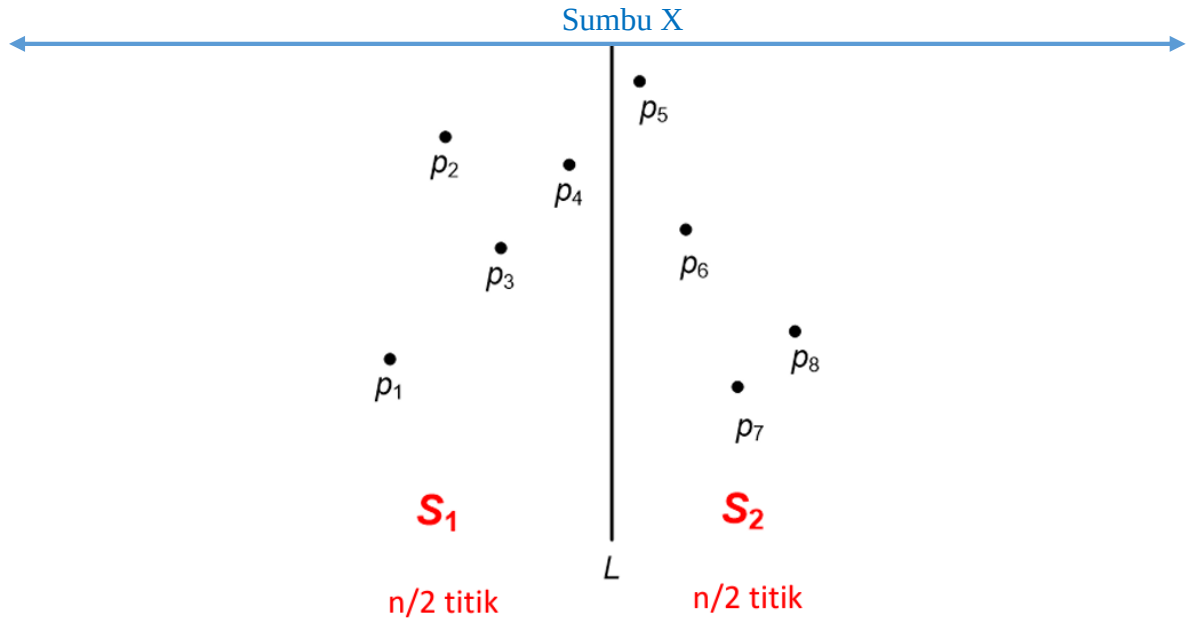
2.2 Algoritme *Divide and Conquer* dalam Menyelesaikan *Closest-Pair Problem*

Berbeda dengan algoritme "membabi buta" seperti *brute force*, *divide and conquer* membagi permasalahan yang harus ia selesaikan menjadi beberapa bagian yang berukuran sama. Oleh karena perlu dipanggil berulang-ulang, algoritme ini dibuat sebagai sebuah fungsi rekursif bernama `closest_pair`, menerima parameter sebuah larik titik (matriks) yang sudah terurut menaik pada absisnya (`points_list`), serta jumlah titik (`num`).

Pada proses pengurutan, algoritme yang digunakan adalah *quicksort*, yang juga merupakan *divide and conquer sorting*. Namun, algoritme tersebut tidak akan dijelaskan lebih lanjut karena tidak termasuk dalam cakupan tugas ini. Seperti yang sudah dicantumkan, tujuan *sorting* adalah mendapatkan larik titik yang terurut menaik pada absisnya, demi kemudahan membagi larik tersebut menjadi dua bagian, seperti pada **Gambar 2.2**.

Algoritme *divide and conquer* "murni" untuk *closest-pair* memang hanya bisa dilakukan jika masukan berukuran 2^k, dengan k sembarang bilangan bulat positif, sebab proses "DIVIDE" selalu membagi dua ukuran larik. Jadi, hal pertama yang dilakukan adalah memeriksa `num`. Ada dua basis yang dibuat, yaitu untuk jumlah titik adalah 3 atau 2. Di sini, mau tidak mau metode *brute force* harus dijalankan. Untuk urusan inilah, algoritme *brute force* pun terpaksa diimplementasikan dalam bentuk fungsi bernama `closest_pair_BF`. Pada basis, jarak antara kedua/ketiga titik tersebut langsung dihitung dan dijadikan sebagai jarak minimum (tahap "CONQUER"). Jarak minimum di sini belum tentu merupakan solusi akhir, sebab bisa jadi harus dibandingkan dengan jarak minimum dari proses lain pada saat "COMBINE".

Kondisi rekurens adalah saat jumlah titik lebih dari tiga. Ini dianggap ukuran yang masih belum cukup kecil untuk menghitung jarak. Persoalan akan langsung dibagi menjadi dua bagian dengan memanggil fungsi `closest_pair` sebanyak dua kali; satu untuk setengah pertama larik titik, dan satu lagi untuk sisanya, dengan parameter `num` masing-masing `num`/2 (atau `num`//2 dalam Python untuk memastikan angkanya bulat).

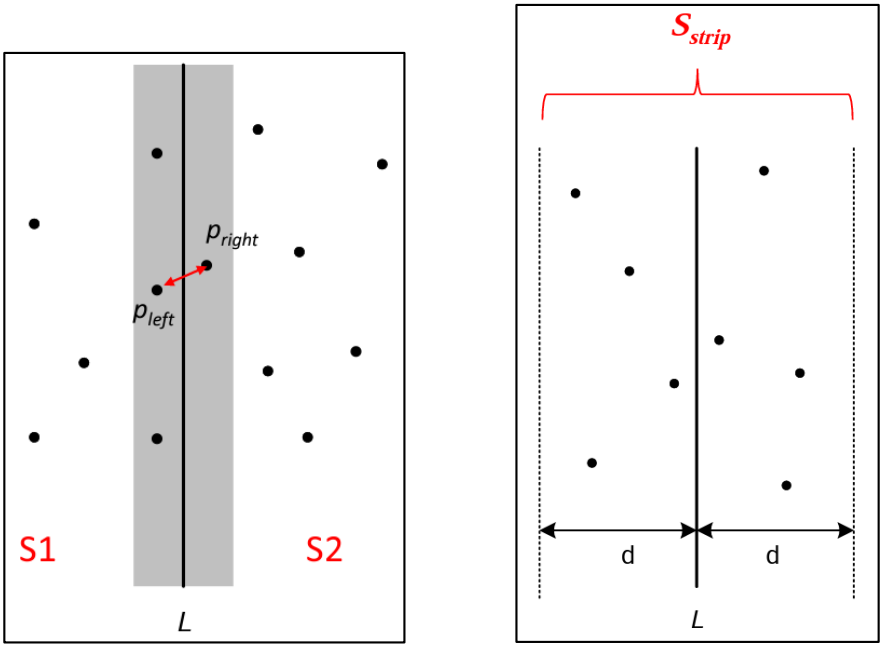


Gambar 2.2 Ilustrasi Pembagian Larik Titik Menjadi Dua Berdasarkan Absis
(Dimodifikasi dari: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf)

Pada kondisi rekurens, terdapat hal khusus yang harus diperiksa. Tidak boleh terlupa, bahwa masih ada kemungkinan, dua titik yang terdekat terpisahkan oleh garis imajiner pembatas. Contohnya dalam **Gambar 2.2**, p_4 dan p_5 adalah pasangan solusi (asumsi saja). Dikutip dari sumber [2], jika ada pasangan titik p_{kiri} and p_{kanan} yang jaraknya lebih kecil dari jarak minimum sementara (d), maka kasusnya adalah:

a. Absis dari p_{kiri} dan p_{kanan} berbeda paling banyak sebesar d.
b. Ordinat dari p_{kiri} dan p_{kanan} berbeda paling banyak sebesar d.

Ini berarti p_{kiri} dan p_{kanan} adalah sepasang titik yang berada di daerah sekitar garis vertikal L (daerah abu-abu), yang memiliki lebar $2d$ dan membelah sumbu X menjadi dua bagian besar. Daerah itu selanjutnya disebut "strip".



Gambar 2.3 Ilustrasi "Strip"
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf)

Dalam kasus tiga dimensi (seperti yang diharapkan dari tugas ini), pemisah p_{kiri} dan p_{kanan} bukanlah sebatas garis, melainkan sebuah bidang YZ berbentuk "persegi panjang" (panjangnya membentang di sumbu Y dan lebarnya membentang di sumbu Z, atau sebaliknya). Bidang tersebut akan membelah sumbu X sehingga terbentuk dua "balok" besar. Strip pun jadi berbentuk 3 dimensi, yaitu balok. Untuk menangani kasus ini, diberikan tambahan satu kasus oleh penulis: c. Nilai pada sumbu Z dari p_{kiri} dan p_{kanan} berbeda paling banyak sebesar d. Hal ini masih masuk akal sebab ada tiga sumbu yang harus diperhatikan.

Kembali ke cerita awal, p_{kiri} yang memenuhi syarat masuk strip dimasukkan ke larik tambahan satu, sedangkan p_{kanan} yang memenuhi syarat masuk strip dimasukkan ke larik tambahan dua. Dua buah kalang masing-masing ditugaskan untuk mengiterasi kedua larik tambahan, dan menghitung jarak dua titik yang ditunjuk, apabila memenuhi salah satu:

a. $|x_{p_1} - x_{p_2}| > d$
b. $|y_{p_1} - y_{p_2}| > d$
c. $|z_{p_1} - z_{p_2}| > d$,

dan sudah pasti hasil hitungan tersebut, jika nilainya lebih kecil, akan menggantikan jarak minimum sementara.

Proses rekurens akan diulang terus sampai berhenti pada salah satu basis. Jika diperhatikan lebih lanjut, dapat ditentukan kompleksitas algoritme dari keseluruhan proses, yaitu:

$$T(n) = \begin{cases} a, & n = 2 \\ 2T\left(\frac{n}{2}\right) + cn, & n > 2 \end{cases}$$

Solusi persamaan di atas menggunakan Teorema Master menghasilkan T(n) = O(n log n), yang jauh lebih baik dari milik algoritme *brute force*.

2.3 Fungsi-Fungsi Pendukung

Modularitas adalah kunci keterbacaan dan efektivitas sebuah program. Maka dari itu, penting untuk membagi program utama menjadi beberapa fungsi kecil yang memiliki peran masing-masing. Dalam menyelesaiakan permasalahan pasangan titik terdekat ini, penulis membuat berbagai fungsi yang dapat dipanggil berkali-kali saat dibutuhkan.

```
#DEKLARASI FUNGSI DAN PROSEDUR
# calculate_distance(point1, point2)
# partition(points_list, start, end, axis)
# quick_sort(points_list, start, end, axis)
# closest_pair(points_list, num)
# closest_pair_generalized(points_list, num, dimension)
# closest_pair_BF(points_list, n)
# print3D(points_list, min_dist_list)
```

Gambar 2.4 Fungsi-Fungsi Pendukung dalam FUNCTIONS.py
(Sumber: Dokumentasi penulis)

Berikut adalah deskripsi singkat dari masing-masing fungsi:

1. **calculate_distance** : menghitung jarak euclidean antara dua titik. Kedua titik tersebut dapat berdimensi berapa pun.
2. **partition** : mempartisi senarai berisi koordinat titik menjadi dua bagian berdasarkan *axis* (absis atau ordinat). *Axis* 0 berarti sumbu X, dan 1 berarti sumbu Y.
3. **quick_sort** : mengurutkan senarai berisi koordinat titik berdasarkan *axis* (absis atau ordinat) menggunakan algoritme *divide and conquer*: *quicksort*.
4. **closest_pair** : mencari pasangan-pasangan titik acak dalam larik, yang jaraknya terdekat, dengan menggunakan algoritme *divide and conquer*. Prasyarat: larik titik sudah dalam keadaaan terurut menaik pada absisnya, dan kedua titik berdimensi tiga.
5. **closest_pair_generalized** : mencari pasangan-pasangan titik acak dalam larik, yang jaraknya terdekat, dengan menggunakan algoritme *divide and conquer*. Prasyarat: larik harus sudah dalam keadaaan terurut menaik pada absisnya, dan kedua titik dapat berdimensi berapa pun, tergantung parameter "dimension".
6. **closest_pair_BF** : mencari pasangan-pasangan titik acak dalam larik yang jaraknya terdekat dengan menggunakan algoritme *brute force*; dimensi bebas.
7. **print3D** : mencetak titik-titik dalam bentuk gambar koordinat Cartesius tiga dimensi, dengan bantuan *library* "Matplotlib".

Implementasi lengkap dari fungsi-fungsi atau prosedur pendukung tersebut bisa dilihat dalam “FUNCTIONS.py” () yang juga terdapat pada pranala di bagian **LAMPIRAN**.

2.4 Program Utama

Sebuah program tidak akan bisa berjalan tanpa “pusat” yang mengatur segalanya. Berikut adalah program utama dari *closest-pair problem solver* yang ditampilkan per bagian:

```
#NIM: 13521140
#Nama: Ryan Samuel Chandra
#Mata Kuliah: IF2211 Strategi Algoritma
#Topik: Tugas Kecil 2
#Tanggal: Jumat, 24 Februari 2023

#Program CLOSEST_PAIR
#Spesifikasi : Program untuk mencari pasangan-pasangan titik acak yang jaraknya terdekat dalam sumbu
#              tiga dimensi, menggunakan perhitungan jarak euclidean dengan algoritme brute force,
#              kemudian dibandingkan secara langsung dengan algoritme divide and conquer

#DEKLARASI LIBRARY
import random
import time
from FUNCTIONS import *

#KAMUS GLOBAL
# n : int
# points_list : array[0..n-1] of array[0..2] of int
# min_dist_list : array[0..?] of array[0..1] of array[0..2] of int
# min_dist_list2 : array[0..?] of array[0..1] of array[0..2] of int
# point : array[0..2] of int (pencacah list of list)
# i : int (pencacah)
# minimum, minimum2 : float
# num_of_calc, num_of_calc2 : int
# start_time1, time1 : time (dari library)
# start_time2, time2 : time (dari library)
# question : char


#TAMPILAN AWAL
print("\n")
print("
print("
print("
print("
print("
print("
print("\n")


#INISIALISASI
n = 0
while (n < 2):
    n = int(input("Masukkan jumlah titik: "))
    if (n < 2):
        print("Masukan harus lebih besar dari 1!")
points_list = [[(round(random.uniform(-50.00,50.00), 2)) for j in range(3)] for i in range(n)]
```

```
#INISIALISASI
n = 0
while (n < 2):
    n = int(input("Masukkan jumlah titik: "))
    if (n < 2):
        print("Masukan harus lebih besar dari 1!")
points_list = [[(round(random.uniform(-50.00,50.00), 2)) for j in range(3)] for i in range(n)]


#PROGRAM UTAMA
question = 'x'
while (question != '0' and question != '1'):
    question = str(input("Apakah Anda ingin mencetak titik-titik Anda? (0/1): "))
    if (question != '0' and question != '1'):
        print("Masukkan '0' untuk tidak, atau '1' untuk ya")

if (question == '1'):
    print("\nTitik:")
    for point in (points_list):
        print(point)

# ------------------------BRUTE FORCE------------------------
start_time1 = time.time()
minimum, num_of_calc, min_dist_list = closest_pair_BF(points_list, n)
time1 = time.time() - start_time1
# -----------------------------------------------------------

# --------------------DIVIDE AND CONQUER---------------------
quick_sort(points_list, 0, len(points_list)-1, 0)
start_time2 = time.time()
minimum2, num_of_calc2, min_dist_list2 = closest_pair(points_list, n)
time2 = time.time() - start_time2
# -----------------------------------------------------------
```

```
#MENCETAK KE LAYAR
print("\n================================================")
print("                    BRUTE FORCE                  ")
print("================================================")
print("Jarak dua titik terdekat: %.5f satuan," % minimum)
print("yaitu antara titik: ", end='')
for i in range(len(min_dist_list)):
    print("\n", min_dist_list[i][0], "dan", min_dist_list[i][1])

print("\nWaktu komputasi: %.10f detik" % (time1))
print("Jumlah perhitungan = {0}({1}-1)/2 = {2}".format(n, n, num_of_calc))


print("\n================================================")
print("                DIVIDE AND CONQUER               ")
print("================================================")
print("Jarak dua titik terdekat: %.5f satuan," % minimum2)
print("yaitu antara titik: ", end='')
for i in range(len(min_dist_list2)):
    print("\n", min_dist_list2[i][0], "dan", min_dist_list2[i][1])

print("\nWaktu komputasi: %.10f detik" % (time2))
print("Jumlah perhitungan =", num_of_calc2)
```

```
#TAMBAHAN PENGGAMBARAN 3D
question = 'x'
while (question != '0' and question != '1'):
    question = str(input("\nApakah Anda ingin menampilkan gambar 3D? (0/1): "))
    if (question != '0' and question != '1'):
        print("Masukkan '0' untuk tidak, atau '1' untuk ya")

if (question == '1'):
    print3D(points_list, min_dist_list)
    print("\nTerima kasih sudah menggunakan layanan ini :)")
else:
    print("Terima kasih sudah menggunakan layanan ini :)")
```

Gambar 2.5 *Source Code* Program Utama Bagian per Bagian
(Sumber: dokumentasi penulis, dari fail: CLOSEST_PAIR.py)

Agar tidak melewatkan inti dari tugas ini, penulis memberikan dokumentasi tambahan:

```
def closest_pair(points_list, num):
    #Mencari pasangan-pasangan titik acak dalam points_list yang jaraknya terdekat
    #dengan menggunakan algoritme divide and conquer. Prasyarat: points_list harus
    #sudah dalam keadaaan terurut menaik pada absisnya; titik berdimensi tiga
#ALGORITME
    min_dist_list = []
    if (num == 3):
        minimum, num_of_calc, min_dist_list = closest_pair_BF(points_list, num)
    elif (num == 2):
        minimum = calculate_distance(points_list[0], points_list[1])
        num_of_calc = 1
        min_dist_list.append([points_list[0], points_list[1]])
    else:
        points_list1 = []
        points_list2 = []

        for i in range(0, num//2, 1):
            points_list1.append(points_list[i])
        for i in range(num//2, num, 1):
            points_list2.append(points_list[i])

        minimum1, num_of_calc1, min_dist_list1 = closest_pair(points_list1, num//2)
        minimum2, num_of_calc2, min_dist_list2 = closest_pair(points_list2, num//2)
        num_of_calc = num_of_calc1 + num_of_calc2

        if (minimum1 < minimum2):
            minimum = minimum1
            min_dist_list = min_dist_list1
        elif (minimum2 < minimum1):
            minimum = minimum2
            min_dist_list = min_dist_list2
        else: #minimum1=minimum2
            minimum = minimum1
            min_dist_list = min_dist_list1 + min_dist_list2

        #Perhitungan dalam "STRIP"
        additional_list1 = []
        additional_list2 = []
        for point in points_list1:
            if (point[0] >= points_list[num//2 - 1][0] - minimum):
                additional_list1.append(point)
        for point in points_list2:
            if (point[0] <= points_list[num//2 - 1][0] + minimum):
                additional_list2.append(point)
```

```
        for i in range(0, len(additional_list1), 1):
            for j in range(0, len(additional_list2), 1):
                if (abs(additional_list1[i][0] - additional_list2[j][0]) > minimum
                or abs(additional_list1[i][1] - additional_list2[j][1]) > minimum
                or abs(additional_list1[i][2] - additional_list2[j][2]) > minimum):
                    pass #tidak diproses
                else:
                    temp_dist = calculate_distance(additional_list1[i], additional_list2[j])
                    num_of_calc = num_of_calc + 1
                    if (temp_dist < minimum):
                        minimum = temp_dist
                        min_dist_list = []
                        min_dist_list.append([additional_list1[i], additional_list2[j]])
                    elif (temp_dist == minimum):
                        min_dist_list.append([additional_list1[i], additional_list2[j]])

    return (minimum, num_of_calc, min_dist_list)
```

Gambar 2.6 Fungsi *Divide and Conquer* Mencari Pasangan Titik Terdekat
(Sumber: dokumentasi penulis, dari fail: FUNCTIONS.py)

2.5 Generalisasi Program

Sebuah titik tidak melulu terletak pada bidang tiga dimensi, melainkan bisa di sembarang dimensi. Titik-titik tersebut bisa membentuk vektor yang memiliki besar dan arah. Jika ada sekumpulan vektor di R^N, dengan N adalah dimensi, setiap vektor dapat dinyatakan dalam bentuk $\mathbf{x} = (x_1, x_2, \ldots x_N)$, $\mathbf{y} = (y_1, y_2, \ldots y_N)$. Rumus yang digunakan untuk mencari jarak kedua titik hanya merupakan modifikasi dari rumus sebelumnya:

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_N - y_N)^2}$$

Dengan demikian, generalisasi dapat dilakukan dengan sedikit mengubah program utama pada bagian masukan (ditambahkan dimensi $\geqslant 1$), serta melengkapi fungsi `closest_pair` dengan kalang [0..*N*] pada setiap operasi yang melibatkan sumbu koordinat; menjadi sebuah fungsi baru: `closest_pair_generalized(points_list, num, dimension)`.

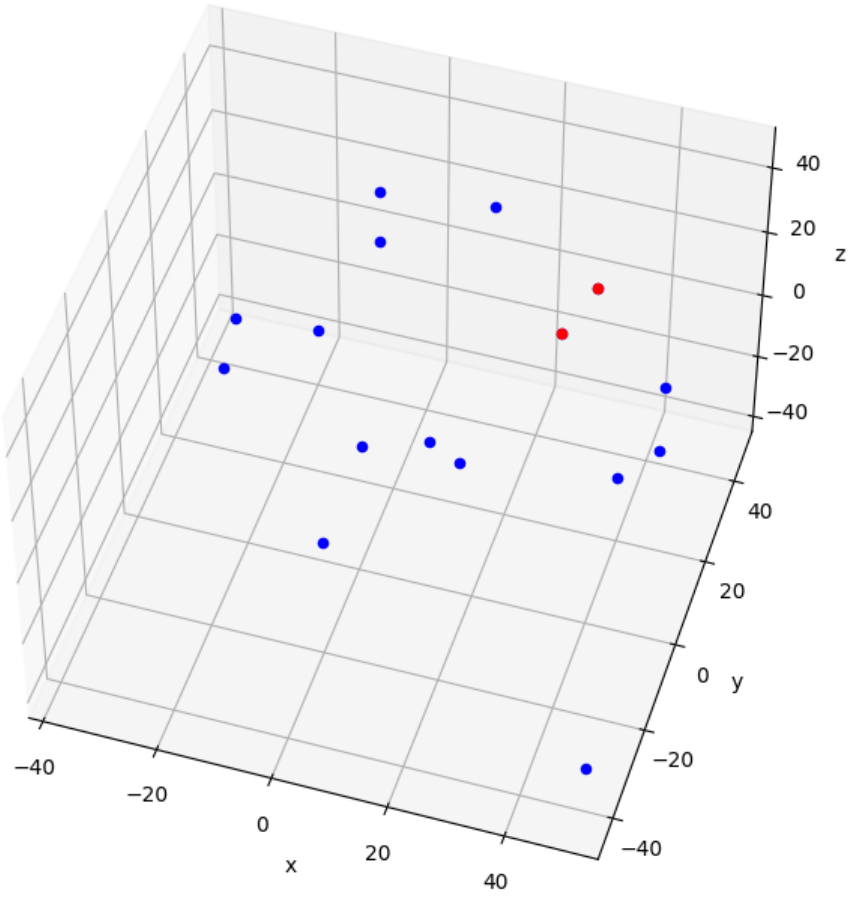
Sayangnya, penggambaran fisik bangun empat dimensi atau lebih belum dapat dilakukan oleh manusia sampai saat ini. Maka dari itu, penulis hanya membuat program penggambaran khusus untuk tiga dimensi pada fail CLOSEST_PAIR.py. Sedangkan implementasi dari generalisasi program itu sendiri dapat dilihat secara lengkap tanpa penggambaran pada fail CLOSEST_PAIR_GENERALIZED.py.

```
#INISIALISASI
N = 0
n = 0

while (N < 1):
    N = int(input("Masukkan dimensi: "))
    if (N < 1):
        print("Masukan harus lebih besar dari 0")
while (n < 2):
    n = int(input("Masukkan jumlah titik: "))
    if (n < 2):
        print("Masukan harus lebih besar dari 1")
points_list = [[(round(random.uniform(-50.00,50.00), 2)) for j in range(N)] for i in range(n)]
```

Gambar 2.7 Perubahan pada Program Utama untuk Generalisasi
(Sumber: dokumentasi penulis)

BAB 3
DOKUMENTASI EKSPERIMEN


```
Masukkan jumlah titik: 16
Apakah Anda ingin mencetak titik-titik Anda? (0/1): 1

Titik:
[9.61, 9.56, -11.08]
[47.97, -15.34, 27.68]
[-24.23, 2.67, 7.25]
[-10.5, 46.19, -1.38]
[43.29, 35.2, -33.83]
[45.87, 25.63, 0.9]
[18.03, -1.66, 1.04]
[36.88, 11.89, 46.03]
[8.24, -24.5, 32.63]
[15.52, 26.23, 44.21]
[7.57, -42.64, 27.08]
[-6.86, 34.86, 29.31]
[33.07, 4.64, 39.78]
[-36.23, 43.19, -33.32]
[49.61, -34.96, -38.89]
[-3.41, -9.58, 43.48]

================================================
                  BRUTE FORCE
================================================
Jarak dua titik terdekat: 10.30248 satuan,
yaitu antara titik:
 [36.88, 11.89, 46.03] dan [33.07, 4.64, 39.78]

Waktu komputasi: 0.0000000000 detik
Jumlah perhitungan = 16(16-1)/2 = 120

================================================
               DIVIDE AND CONQUER
================================================
Jarak dua titik terdekat: 10.30248 satuan,
yaitu antara titik:
 [33.07, 4.64, 39.78] dan [36.88, 11.89, 46.03]

Waktu komputasi: 0.0000000000 detik
Jumlah perhitungan = 12

Apakah Anda ingin menampilkan gambar 3D? (0/1): 1
Mohon menunggu pemrosesan gambar...
```



Gambar 3.1 Kasus jumlah titik adalah 16,
koordinat titik ditampilkan, gambar 3D ditampilkan

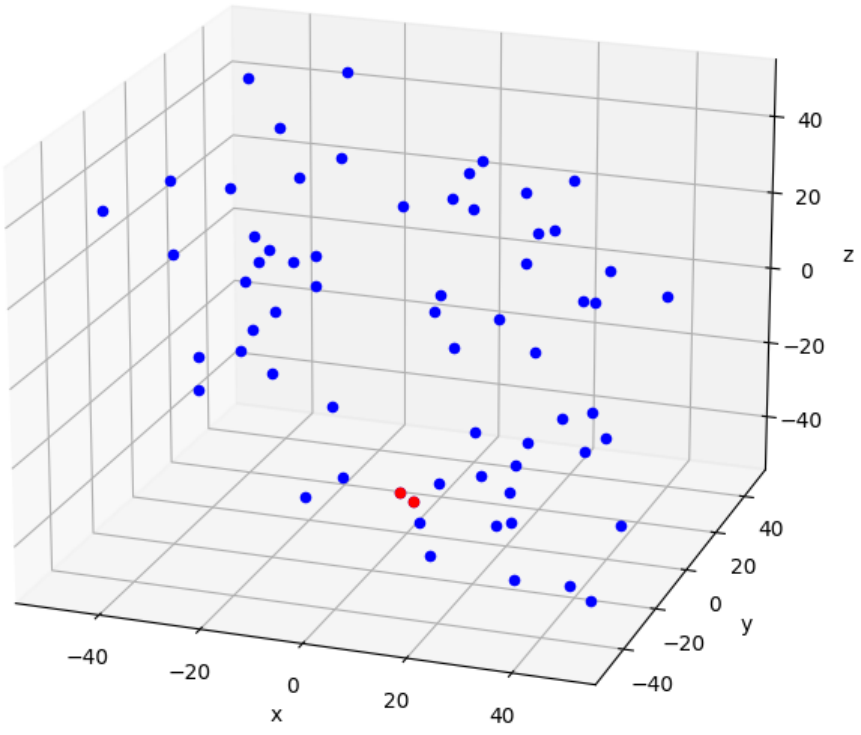
(Fail: CLOSEST_PAIR.py)

Semua titik digambarkan dengan warna biru, kecuali titik yang menjadi pasangan solusi; digambarkan dengan warna merah. Secara visual pada gambar di atas, dua titik berwarna merah mungkin bukan yang paling dekat. Hal ini dikarenakan gambar tiga dimensi dalam tampilan dua dimensi tidak memungkinkan pembaca untuk melihat dari sudut pandang yang berbeda. Padahal, memang kedua titik tersebutlah yang memiliki jarak terdekat. Meskipun demikian, penulis tetap berusaha memperlihatkan sudut pandang terbaik.

```
 ██████╗██╗      ██████╗ ███████╗███████╗███████╗████████╗      ██████╗  █████╗ ██╗██████╗
██╔════╝██║     ██╔═══██╗██╔════╝██╔════╝██╔════╝╚══██╔══╝      ██╔══██╗██╔══██╗██║██╔══██╗
██║     ██║     ██║   ██║███████╗█████╗  ███████╗   ██║         ██████╔╝███████║██║██████╔╝
██║     ██║     ██║   ██║╚════██║██╔══╝  ╚════██║   ██║         ██╔═══╝ ██╔══██║██║██╔══██╗
╚██████╗███████╗╚██████╔╝███████║███████╗███████║   ██║         ██║     ██║  ██║██║██║  ██║
 ╚═════╝╚══════╝ ╚═════╝ ╚══════╝╚══════╝╚══════╝   ╚═╝         ╚═╝     ╚═╝  ╚═╝╚═╝╚═╝  ╚═╝

Masukkan jumlah titik: 64
Apakah Anda ingin mencetak titik-titik Anda? (0/1): 0

================================================
                  BRUTE FORCE
================================================
Jarak dua titik terdekat: 3.30232 satuan,
yaitu antara titik:
 [1.11, 0.1, -45.89] dan [-1.43, 0.14, -43.78]

Waktu komputasi: 0.0029985905 detik
Jumlah perhitungan = 64(64-1)/2 = 2016

================================================
               DIVIDE AND CONQUER
================================================
Jarak dua titik terdekat: 3.30232 satuan,
yaitu antara titik:
 [-1.43, 0.14, -43.78] dan [1.11, 0.1, -45.89]

Waktu komputasi: 0.0009994507 detik
Jumlah perhitungan = 59

Apakah Anda ingin menampilkan gambar 3D? (0/1): a
Masukkan '0' untuk tidak, atau '1' untuk ya

Apakah Anda ingin menampilkan gambar 3D? (0/1): b
Masukkan '0' untuk tidak, atau '1' untuk ya

Apakah Anda ingin menampilkan gambar 3D? (0/1): 4
Masukkan '0' untuk tidak, atau '1' untuk ya

Apakah Anda ingin menampilkan gambar 3D? (0/1): 1
Mohon menunggu pemrosesan gambar...
```

Gambar 3.2 Kasus jumlah titik adalah 64, koordinat titik tidak ditampilkan, ada kesalahan input, gambar 3D ditampilkan

(Fail: CLOSEST_PAIR.py)



Gambar 3.3 Kasus jumlah titik adalah 128, terdapat kesalahan input, koordinat titik tidak ditampilkan, gambar 3D tidak ditampilkan

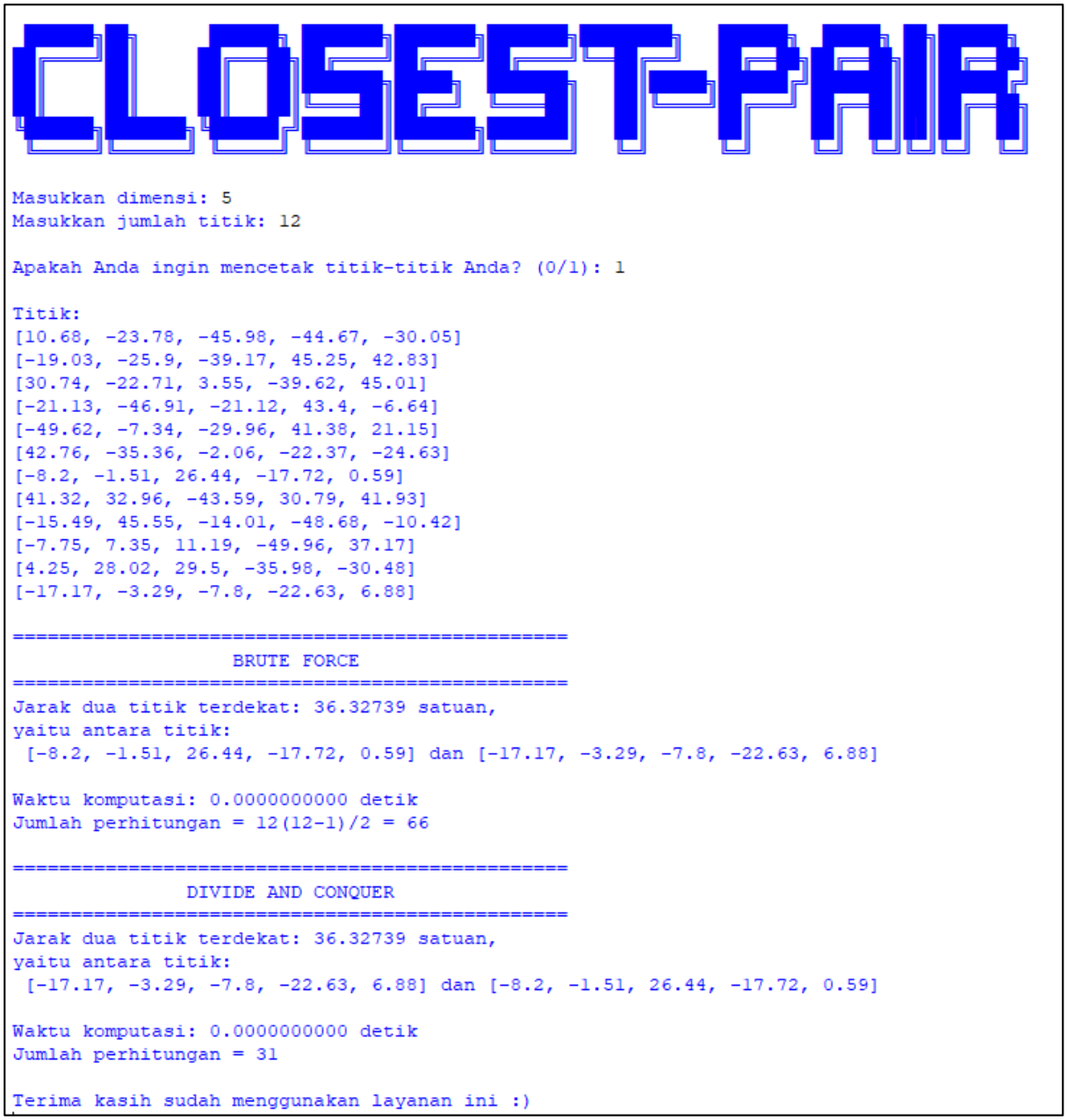
(Fail: CLOSEST_PAIR.py)

Gambar 3.4 Kasus jumlah titik adalah 1000, koordinat titik tidak ditampilkan, gambar 3D tidak ditampilkan

(Fail: CLOSEST_PAIR.py)

Gambar 3.5 Kasus dimensi = 4, jumlah titik = 1024, koordinat titik tidak ditampilkan

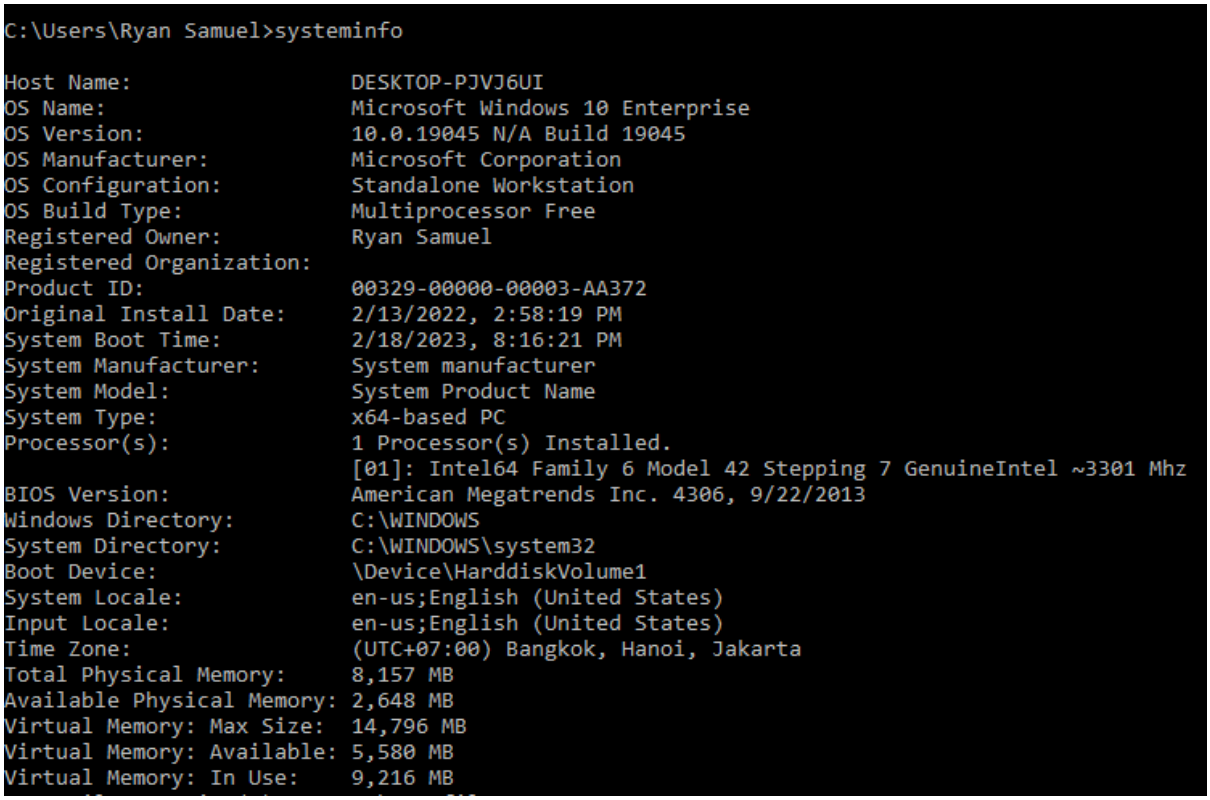
(Fail: CLOSEST_PAIR_GENERALIZED.py)



Gambar 3.6 Kasus dimensi = 5, jumlah titik = 10, koordinat titik ditampilkan

(Fail: CLOSEST_PAIR_GENERALIZED.py)

Semua solusi di atas, diciptakan dalam sistem operasi Windows, dengan spesifikasi komputer yang digunakan sebagai berikut:



Gambar 3.7 Spesifikasi Komputer
(Sumber: dokumentasi penulis)

BAB 4
PENUTUP

4.1 Kesimpulan

Closest-pair problem, atau permasalahan mencari pasangan titik terdekat, adalah salah satu persoalan terdahulu pada geometri komputasional. Masalahnya adalah: diberikan himpunan *Q* yang berisi $n \geq 2$ buah titik berdimensi tertentu, untuk kemudian dicari pasangan mana yang memiliki jarak Euclidean terdekat.

Algoritme *divide and conquer* adalah algoritme yang memiliki dua bagian utama: *divide* berarti membagi persoalan menjadi beberapa upapersoalan yang memiliki kemiripan karakteristik dengan persoalan semula, namun ukurannya lebih kecil. Sedangkan *conquer* berarti menyelesaikan masing-masing upapersoalan secara langsung jika sudah berukuran kecil. Semua solusi dari upapersoalan akan di-*combine* agar solusi terbaik dapat ditentukan.

Algoritme *divide and conquer* dalam menyelesaikan *closest-pair problem* dinilai lebih efektif dibandingkan *brute force*. Dengan jumlah titik dan dimensi yang sama, D&C berhasil menyelesaikan pekerjaan dengan lebih cepat, jumlah operasi perhitungan yang lebih sedikit, serta solusi yang 93% akurat. Program lengkap dapat dilihat pada pranala di bagian **LAMPIRAN**, sedangkan contoh eksekusi ada di bagian **BAB 3**.

4.2 Saran

1. Sebaiknya, penyelesaian masalah titik terdekat dilakukan dalam bahasa pemrograman yang mampu mengoperasikan *array* (larik) dengan perintah mudah, seperti yang dilakukan pada tugas ini. Penggunaan larik sangat banyak dan penting untuk kasus ini.
2. Dalam mengerjakan pemrograman, sebaiknya pemrogram memiliki catatan kecil tentang fungsi, prosedur, dan juga fitur yang telah dibuat. Hal ini bertujuan untuk mempermudah modifikasi program, serta agar tidak ada duplikat fungsi [1].
3. Membaca spesifikasi dengan teliti sampai tuntas sebaiknya dilakukan sebelum memulai pemrograman. Dengan demikian, tidak perlu ada perubahan massal di tengah pekerjaan karena ketidaksesuaian dengan spesifikasi [1].
4. Seharusnya, durasi pengerjaan tugas diberikan lebih dari satu minggu, mengingat tugas "bermain" algoritme seperti ini membutuhkan eksplorasi yang sangat banyak, serta mungkin dapat dikembangkan lebih lanjut apabila waktu tersedia.

4.3 Refleksi

Tugas kecil kedua memberikan setidaknya banyak ilmu baru dalam pemrograman dengan bahasa Python. Meskipun pengalaman saya dalam mengerjakan tugas ini tidak lebih baik dari tugas kecil sebelumnya, saya belajar bahwa pengaturan waktu dan koordinasi yang baik sangat diperlukan bahkan saat tidak bekerja dalam tim [1]. Apabila satu baris kode saja tidak mengembalikan nilai yang benar, maka keseluruhan program menjadi salah.

Dengan keterbatasan pengetahuan dan keahlian, saya pun jadi harus berusaha sangat keras mengeksplor lebih dan lebih dalam lagi mengenai tugas ini, baik dari internet, teman, juga saudara. Akhirnya, usaha saya tidak sia-sia. Tugas pemrograman kesekian kalinya di semester 4 terselesaikan dengan baik dan tepat waktu.

LAMPIRAN

https://github.com/RyanSC06/Tucil2_13521140.git

Tabel A Runut Pengerjaan Tugas

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	☑	

DAFTAR REFERENSI

1. Samuel, R. Bandung, 24 Januari 2023. *Implementasi Algoritme Brute Force dalam Bahasa Pemrograman C untuk Menyelesaikan Permainan Kartu 24*. (Diakses tanggal 28 Februari 2023 dari https://github.com/RyanSC06/Tucil1_13521140/tree/main/doc)
2. Munir, Rinaldi. 2021. *Algoritma Divide and Conquer (Bagian 2)*. (Dikases pada tanggal 28 Februari 2023 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf)
3. Munir, Rinaldi. 2021. *Algoritma Divide and Conquer (Bagian 1)*. (Dikases pada tanggal 28 Februari 2023 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf)
4. Munir, Rinaldi. 2022. *Algoritma Brute Force (Bagian 1)*. (Diakses tanggal 28 Februari 2023 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf)
5. Greg, Bang. 29 Mei 2017. *Cara Menentukan Jarak Dua Titik Pada Bidang Koordinat Cartesius*. (Diakses dari https://www.tipsbelajarmatematika.com/2017/05/cara-menentukan-jarak-dua-titik-pada.html pada tanggal 28 Februari 2023)
6. Yudistira, Reno. Tanpa tanggal. *Jarak Titik, Garis, dan Bidang*. (Diakses dari https://renoyudistira1412.wordpress.com/jarak-titik-garis-dan-bidang/ pada tanggal 28 Februari 2023)
7. Ge Q., Wang H-T., dan Zhu H. 22 Agustus 2005. *An Improved Algorithm for Finding the Closest Pair of Points*. Journal of Computer Science and Technology: January 2006 Vol. 21, No. 1, Hal. 27-31.
8. GeeksforGeeks. 13 Februari 2023. *Closest Pair of Points using Divide and Conquer Algorithm*. (Diakses dari https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/ pada tanggal 28 Februari 2023)