Tugas Kecil 2 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

Mencari Pasangan Titik Terdekat 3D dengan Algoritma *Divide and Conquer*

Disusun oleh:

Mohammad Rifqi Farhansyah

13521166

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2023

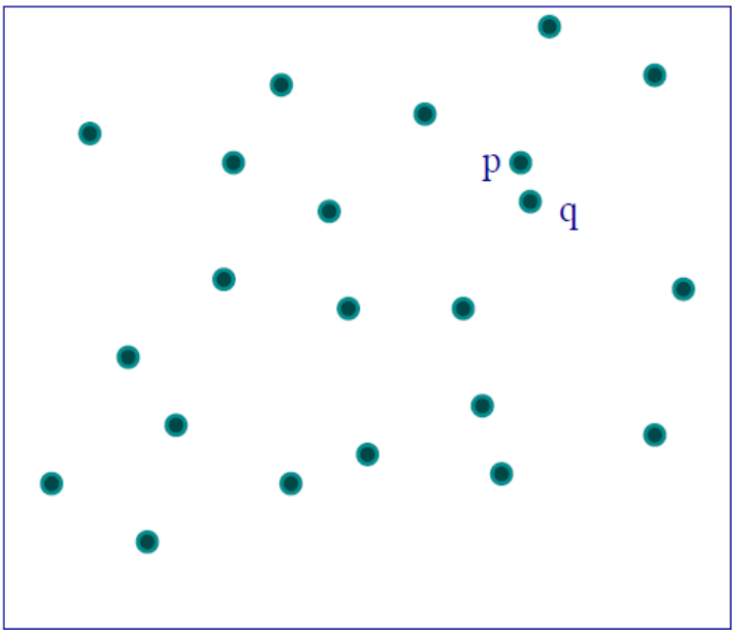
DAFTAR ISI

Bab 1
Deskripsi Masalah

Permasalahan *closest pair* merupakan salah satu permasalahan klasik yang kerap ditemui dalam konteks geometri ruang dan optimasi jarak. Dalam dunia sehari-hari juga banyak ditemui permasalahan semacam ini. Secara singkat permalasahan ini dapat dirumuskan sebagai berikut : "Diberikan n-buah titik dalam *Euclidean Space* dan d dimensi, carilah dua titik dengan jarak terdekat". Pada laporan ini, penulis akan menjelaskan sebuah solusi atas permasalahan tersebut (pencarian dua titik terdekat pada *Euclidean Space*) dengan menggunakan algoritma *Divide and Conquer*. Secara naif, permasalahan ini dapat diselesaikan dengan menggunakan algoritma *brute force* yang membutuhkan kompleksitas waktu $O(N^2)$. Dalam kassus N yang besar (N > 100000), komputasi dengan algoritma *brute force* akan memakan waktu yang cukup lama, bahkan dengan perkembangan teknologi komputer secanggih saat ini. Untuk itu, diperkenalkan metode lain dengan menggunakan algoritma *Divide and Conquer* dalam penyelesaiannya.

Algoritma Divide and Conquer merupakan algoritma yang cukup sering dimanfaatkan dalam menyelesaikan permasalahan di bidang komputasi geometri. Pada dasarnya, algoritma ini bekerja dengan terlebih dahulu memecah sebuah permasalahan menjadi beberapa permasalahan yang lebih kecil. Selanjutnya, tiap sub-permasalahan akan diselesaikan secara terpisah satu sama lain. Proses akan diakhiri dengan penggabungan solusi dari setiap sub-permasalahan untuk mendapatkan solusi global. Algoritma ini memandang penyelesaian sub-permasalahan sebagai suatu set instruksi yang berulang sehingga solusi dapat dicapai dengan pendekatan rekursif.

Masalah pencarian pasangan titik terdekat telah dipelajari secara ekstensif dari masa ke masa, dan beberapa algoritma telah diusulkan untuk menyelesaikannya. Setiap algoritma tersebut memiliki kelebihan dan kekurangan masing-masing, mulai dari algoritma efektif tetapi sulit dalam implementasinya hingga algoritma yang mudah realisasinya tetapi kesangkilannya tergolong cukup rendah. Sementara itu, pada algoritma *divide and conquer* yang disajikan dalam laporan ini, kompleksitas waktu yang dimilikinya adalah $O(n(log(n))^{d-1})$ dengan d sebagai dimensi (lebih baik daripada kompleksitas waktu algoritma *brute force*). Selain itu, implementasi algoritma ini juga tergolong tidak terlalu sulit. Dalam laporan ini, akan diulas lebih lanjut tentang seluruh poin yang disebutkan di atas. Solusi ini terinspirasi oleh sebuah artikel berjudul 'Penyelesaian Masalah Closest Pair dengan Algoritma Divide and Conquer', karya Karol Danutama – 13508040, yang dapat diakses melalui pranala berikut ini.



Gambar 1.1 Ilustrasi Closest Pair Problem
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2010-2011/Makalah2010/MakalahStima2010-055.pdf

Bab 2
Teori Singkat

Landasan teori yang akan digunakan pada penerapan strategi algoritma dalam permasalahan *closest pair*, yaitu: algoritma *Divide and Conquer*. Namun, penulis juga akan membandingkannya dengan solusi menggunakan algoritma *Brute Force*. Algoritma *Divide and Conquer* dan *Brute Force* merupakan salah satu kajian pada mata kuliah IF2211 – Strategi Algoritma.

2.1 Definisi Algoritma *Brute Force*

Brute Force merupakan pendekatan yang lemapang (*straight-forward*) dalam hal pemecahan suatu masalah atau persoalan dengan sangat sederhana, langsung, dan jelas (*obvious-way*). Algoritma *Brute Force* seringkali disebut juga sebagai algoritma naif (*naive algorithm*).

2.2 Karakteristik Algoritma *Brute Force*

Karakteristik algoritma *Brute Force* umumnya tidak mangkus dan sangkil, karena membutuhkan jumlah langkah yang besar dalam penyelesaiannya, sehingga terkadang algoritma *Brute Force* disebut juga sebagai algoritma yang naif. Algoritma *Brute Force* seringkali merupakan pilihan yang kurang disukai karena ketidakmangkusannya itu, tetapi dengan mencari pola-pola yang mendasar, keteraturan, atau trik-trik khusus, biasanya akan membantu kita menemukan algoritma yang lebih cerdas dan lebih mangkus. Berkaitan dengan masalah yang berukuran cukup kecil, kesederhanaan algoritma *Brute Force* biasanya lebih diperhitungkan daripada ketidakmangkusannya. Algoritma *Brute Force* sering digunakan sebagai basis saat membandingkan beberapa alternatif algoritma yang mangkus. Algoritma *Brute Force* seringkali lebih mudah diimplementasikan daripada algoritma-algoritma yang memerlukan penalaran logika lebih mendalam, terkadang algoritma *Brute Force* dapat lebih mangkus (ditinjau dari segi implementasi).

2.3 Keunggulan dan Kekurangan Algoritma *Brute Force*

Algoritma *Brute Force* memiliki beberapa keunggulan dan kekurangan dibandingkan dengan algoritma-algoritma lainnya. Berikut ini merupakan keunggulan dari algoritma *Brute Force*:

- Metode *Brute Force* dapat digunaan untuk memecahkan hampir sebagian besar masalah.
- Metode *Brute Force* cenderung lebih sederhana dan mudah untuk diimplementasikan.
- Metode *Brute Force* menghasilkan algoritma yang layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan string, dan perkalian matriks.
- Metode *Brute Force* menghasilkan algoritma baku untuk tugas-tugas komputasi seperti penjumlahan dan perkalian sebuah bilangan, menentukan elemen minimum atau maksimum dalam sebuah senarai, dan lain-lain.

Sementara itu, kelemahan dari algoritma *Brute Force* antara lain:

- Metode *Brute Force* jarang menghasilkan algoritma yang mangkus dan sangkil.
- Beberapa algoritma *Brute Force* lambat sehingga tidak dapat diterima.
- Tidak sekonstruktif/sekreatif teknik pemecahan lain yang serupa.

2.4 Definisi Algoritma *Divide and Conquer*

Divide and Conquer merupakan algoritma yang didasarkan pada tiga tahapan utama, yaitu: *divide*, *conquer* (*solve*), serta *combine*. *Divide* merupakan upaya untuk membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya berukuran hampir sama). Sementara *conquer* merupayakan tahapan menyelesaikan masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar). Serta *combine* merupakan upaya untuk menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

2.5 Karakteristik Algoritma *Divide and Conquer*

Pada umumnya, teknik *Divide and Conquer* sangat efisien dalam menyelesaikan masalah yang memiliki struktur yang jelas dan teratur. Hal ini karena teknik ini berusaha untuk memecah masalah besar menjadi masalah yang lebih kecil, sehingga masalah yang dipecahkan memiliki ukuran yang lebih kecil dan lebih mudah untuk dipecahkan. Selain itu, teknik *Divide and Conquer* juga dapat digunakan untuk menyelesaikan masalah yang kompleks dan rumit, dengan memecahnya menjadi beberapa masalah yang lebih kecil dan memecahkan setiap masalah kecil secara terpisah. Namun, teknik *Divide and Conquer* tidak selalu cocok untuk semua jenis masalah. Terkadang, proses pemecahan masalah menjadi beberapa masalah kecil dapat menghasilkan banyak *overhead* dan memakan waktu yang lama. Selain itu, teknik ini memerlukan pemilihan yang tepat pada tahap pembagian masalah besar menjadi beberapa masalah kecil, karena pemilihan yang tidak tepat dapat

menghasilkan kinerja yang buruk atau bahkan kesalahan dalam solusi akhir. Dalam prakteknya, teknik *Divide and Conquer* biasanya digunakan dalam algoritma pengurutan data seperti *Merge Sort* dan *Quick Sort*, pencarian dalam data terstruktur seperti *Binary Search*, dan dalam masalah-masalah geometri seperti *Closest Pair*. Karakteristik teknik *Divide and Conquer* yang efektif dalam menyelesaikan masalah yang memiliki struktur dan kompleksitas tertentu menjadikannya salah satu teknik pemecahan masalah yang paling populer dan efektif di dunia komputasi.

2.6 Keunggulan dan Kekurangan Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* memiliki beberapa keunggulan dan kekurangan dibandingkan dengan algoritma-algoritma lainnya. Berikut ini merupakan keunggulan dari algoritma *Divide and Conquer*:

- Algoritma *Divide and Conquer* dapat menghasilkan algoritma yang lebih mangkus dan cepat dibandingkan dengan metode *brute force* pada beberapa masalah yang kompleks..
- Algoritma *Divide and Conquer* dapat dipecah menjadi bagian-bagian yang lebih kecil sehingga lebih mudah untuk diimplementasikan dan dipahami.
- Algoritma *Divide and Conque*r dapat membantu memecahkan masalah-masalah yang terkait dengan pencarian, pengurutan, penggabungan, dan sebagainya.
- Algoritma *Divide and Conquer* memungkinkan untuk melakukan *parallel processing*, yaitu memecahkan bagian-bagian masalah secara bersamaan untuk meningkatkan efisiensi waktu pemrosesan.

Sementara itu, kelemahan dari algoritma *Divide and Conquer*, antara lain:

- Algoritma *Divide and Conquer* memerlukan pengelompokan masalah menjadi bagian-bagian yang lebih kecil, yang bisa memerlukan waktu dan usaha yang signifikan.
- Algoritma *Divide and Conquer* memerlukan penggunaan rekursi yang dalam beberapa kasus dapat menyebabkan penggunaan memori yang lebih besar.
- Algoritma *Divide and Conquer* sering memerlukan analisis matematis yang kompleks untuk memastikan kebenaran dan mangkukannya.

2.7 Landasan Teori Solusi Permasalahan *Closest Pair*

Pencarian dua buah titik terdekat dalam ruang *Euclidean* dengan algoritma *divide and conquer* didasarkan pada prinsip dasar bahwa jarak terdekat antara dua buah titik dapat dicari dengan membagi wilayah yang memuat kedua titik menjadi beberapa sub-wilayah yang lebih kecil. Setiap sub-wilayah ini kemudian diperiksa untuk menemukan jarak terdekat antara dua titik yang mungkin terletak di dalamnya. Jarak antara dua titik pada ruang *Euclidean* berdimensi ‘n’ didefinisikan sebagai berikut.

$$d = \sqrt{\sum_{i}^{n} (p_{1_i} - p_{2_i})^2}$$

Dengan p_1 sebagai titik pertama, p_2 sebagai titik kedua, dan i sebagai indeks sumbu koordinat dari titik. Metode divide and conquer digunakan untuk membagi wilayah menjadi sub-wilayah yang lebih kecil. Pada setiap tahap, wilayah dibagi menjadi dua bagian sejajar dengan salah satu sumbu koordinat. Kemudian, pencarian jarak terdekat dilakukan secara rekursif pada kedua bagian ini. Setelah pencarian pada kedua sub-wilayah selesai, jarak terdekat dari dua titik yang berbeda di seluruh wilayah dipilih. Metode ini menghasilkan kompleksitas waktu yang efisien untuk mencari dua titik terdekat di dalam wilayah dengan ukuran besar. Dengan melakukan pembagian wilayah secara rekursif, kompleksitas waktu dapat dijaga agar menjadi $O(n(\log(n))^{d-1})$, dengan n sebagai jumlah titik yang harus diperiksa dan d sebagai dimensi dari ruang Euclidean yang ditelusuri.

Bab 3
Penerapan Algoritma *Divide and Conquer*

3.1 Algoritma *Brute Force* pada Permasalahan *Closest Pair*

Algoritma *brute force* adalah metode yang digunakan untuk menemukan solusi dengan mencoba semua kemungkinan yang ada. Dalam hal ini, algoritma *brute force* akan menghitung jarak dari seluruh pasangan titik yang ada. Secara garis besar, algoritma *brute force* berjalan seperti berikut:

1. Pilih suatu titik sembarang untuk memulai pencarian dan inisialisasi nilai jarak terdekat
2. Cari titik lain dan periksa apabila pasangan titik belum pernah dicek sebelumnya dan memiliki jarak lebih kecil dari nilai jarak terdekat yang diketahui. Jika iya, perbarui nilai jarak terdekat
3. Lakukan untuk setiap titik pada ruang pencarian

Prosedur pencarian ini cukup simpel namun memakan waktu yang lama untuk jumlah titik yang banyak. Kompleksitas waktu dari algoritma ini adalah $O(n^2)$.

3.2 Algoritma *Divide and Conquer* pada Permasalahan *Closest Pair*

Sementara itu, algoritma divide and conquer membagi permasalahan menjadi subset-subset kecil dengan solusi masing-masing yang kemudian digabungkan menjadi solusi umum. Dalam implementasinya untuk mencari pasangan titik terdekat, algoritma divide and conquer mengikuti tahapan-tahapan berikut:

1. Urutkan titik-titik berdasarkan absis terurut menaik dengan quick sort
2. Bagi himpunan titik menjadi subset kiri dan kanan berdasarkan koordinatnya terhadap suatu garis/bidang maya pembagi, umumnya terletak pada absis median
3. Terdapat tiga kasus terletaknya pasangan titik terdekat: Keduanya terletak pada subset kiri, keduanya terletak pada subset kanan, atau kedua titik terpisah oleh garis/bidang pembagi.
4. Untuk menghadapi kasus ketiga, manfaatkan fakta bahwa jarak titik ke garis pembagi tidaklah lebih besar dari d, dengan d merupakan jarak terkecil dari subset kanan dan kiri. Identifikasikan titik-titik tersebut yang terletak di dalam daerah garis pembagi dan rentangan d. Jika terdapat pasangan titik yang jaraknya lebih kecil dari d, maka untuk setiap selisih koordinat kedua titik, tidaklah lebih besar dari d.
5. Lakukan secara rekursif untuk subset kanan dan kiri.
6. Bandingkan antara ketiga kasus tersebut untuk mendapatkan pasangan titik terdekat.

Algoritma divide and conquer ini merupakan yang paling sangkil dalam menyelesaikan permasalahan pencarian pasangan titik terdekat, dengan kompleksitas waktu $O(n(log(n))^{d-1})$.

3.3 Implementasi Program pada Bahasa Pemrograman Python

Directory Tree dari source code implementasi program adalah sebagai berikut:

```bash
│   README.md
│
├───doc
│       Tucil2_K2_13521166_MohammadRifqiFarhansyah.pdf
│
├───image
│       1000(2).png
│       1000(3).png
│       128(2).png
│       128(3).png
│       16(2).png
│       16(3).png
│       64(2).png
│       64(3).png
│       SS1.png
│       SS2.png
```

```
│   SS3.png
│   SS4.png
│   SS5.png
│
└───src
    │   calculation.py
    │   main.py
    │   visualization.py
    │
    ├───input
    │       input.txt
    │
    ├───output
    │       output.txt
    │
    └───__pycache__
            calculation.cpython-39.pyc
            visualization.cpython-39.pyc
```

Pada laporan ini, program ditulis dalam bahasa pemrograman Python. Bahasa ini dipilih dengan pertimbangan kemudahan visualisasi hasil akhir dengan menggunakan *library* yang tersedia. *Source code* program secara umum terdiri atas satu program utama main.py serta 2 program pendukung yang berisi beberapa fungsi perantara, yaitu visualization.py dan calculation.py, ketiganya terdapat pada folder *'src'*. Program utama berisi seluruh alur program yang telah dirancang sedemikian rupa hingga memiliki efektivitas perhitungan yang cukup tinggi serta tetap memperhatikan aspek ergonomis pada tampilan-nya. File calculation.py berisi seluruh perhitungan yang digunakan dalam memnentukan dua titik terdekat hasil dari masukan random. Sementara file visualization.py berisi fungsi-fungsi yang digunakan untuk mem-visualisasikan point-point tersebut ke dalam sebuah plot tertentu. Selain itu, terdapat pula folder *'doc'* yang berisi laporan Tugas Kecil 2 Strategi Algoritma. Pada folder *'image'* berisi tangkapan layar beberapa fitur utama program. Untuk lebih lengkapnya, source code dapat diakses melalui pranala berikut ini.

3.4 Modul Eksternal

Pada implementasi program digunakan beberapa modul eksternal, seperti *time, matplotlib, typing, math*, dan *random*. Setiap modul memiliki fungsi tersendiri yang akan dimanfaatkan dalam program. Modul *time* merupakan modul yang digunakan untuk waktu saat ini, menghitung waktu lalu, menunggu selama periode waktu tertentu, mengonversi waktu antara format yang berbeda, dan masih banyak lagi. Penggunaan modul *time* adalah saat ingin mengukur waktu eksekusi algoritma *brute force* dan *divide and conquer*. Sementara itu, terdapat modul *matplotlib*. Modul *matplotlib* adalah modul yang digunakan untuk membuat visualisasi data dalam bentuk grafik atau plot. Modul ini menyediakan berbagai jenis plot seperti *scatter plot, line plot, bar plot,* dan masih banyak lagi. Selain itu, modul ini juga memungkinkan kita untuk menyesuaikan berbagai aspek dari plot seperti ukuran, warna, label sumbu, judul, dan legenda. Penggunaan modul *matplotlib* pada program ini adalah dalam memvisualisaikan titik-titik terdekat pada dimensi 2 dan 3. Modul *typing* digunakan untuk menentukan tipe data pada variabel dan fungsi dalam program. Modul ini memungkinkan kita untuk memberikan hint pada tipe data yang diharapkan sehingga dapat memudahkan *debugging* dan meminimalkan kesalahan penulisan kode. Modul *math* digunakan untuk melakukan operasi matematika dalam program. Modul ini menyediakan berbagai fungsi matematika seperti *sin, cos, tan, log, sqrt*, dan lain-lain. Contoh penggunaan modul *math* pada program ini adalah perhitungan *euclidean distance*. Serta modul yang terakhir adalah modul *random*. Modul *random* digunakan untuk menghasilkan nilai acak dalam program. Modul ini menyediakan berbagai fungsi untuk menghasilkan nilai acak seperti *randint, random, uniform*, dan lain-lain. Contoh penggunaan modul *random* pada program ini adalah menghasilkan nilai acak untuk koordinat tiap *point* atu titik.

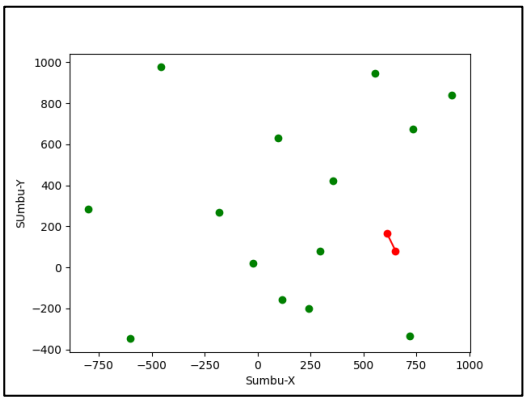
3.5 Testing dan Hasil Tangkapan Layar Input dan Output

Ujicoba dilakukan pada spesifikasi laptop sebagai berikut:

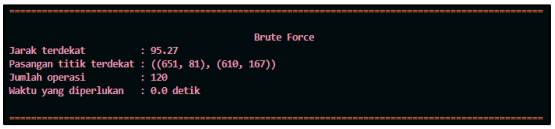
Operating System	:	Windows 10 Home Single Language 64-bit (10.0, Build 19042)
System Model	:	VivoBook_ASUSLaptopX421EPY
Processor	:	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz (8 CPUs), ~2.8GHz
Memory	:	8192MB RAM
Page file	:	21504MB used, 3923MB available
DirectX version	:	DirectX 12

- Ujicoba n = 16 pada 2 dimensi

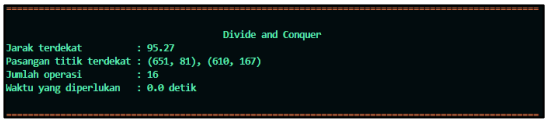
Gambar 3.1.1 Informasi Masukan
Sumber: Dokumen Penulis



Gambar 3.1.2 Visualisasi Titik-Titik
Sumber:Dokumen Penulis

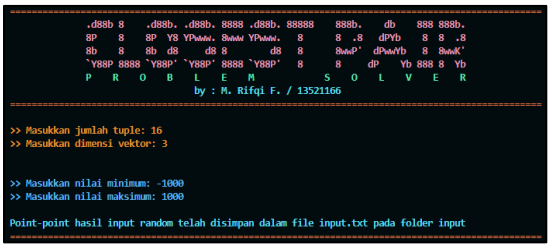


Gambar 3.1.3 Hasil Brute Force
Sumber: Dokumen Penulis

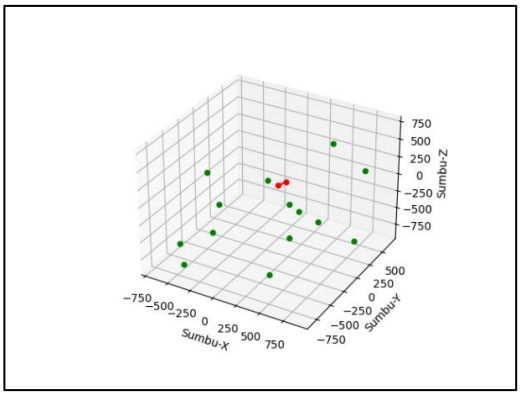


Gambar 3.1.4 Hasil Divide And Conquer
Sumber:Dokumen Penulis

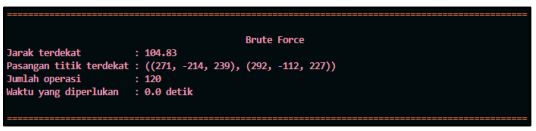
- Ujicoba n = 16 pada 3 dimensi



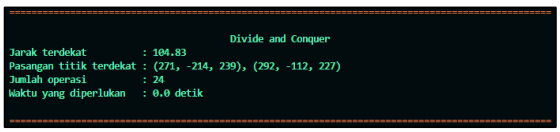
Gambar 3.2.1 Informasi Masukan
Sumber: Dokumen Penulis



Gambar 3.2.2 Visualisasi Titik-Titik
Sumber:Dokumen Penulis



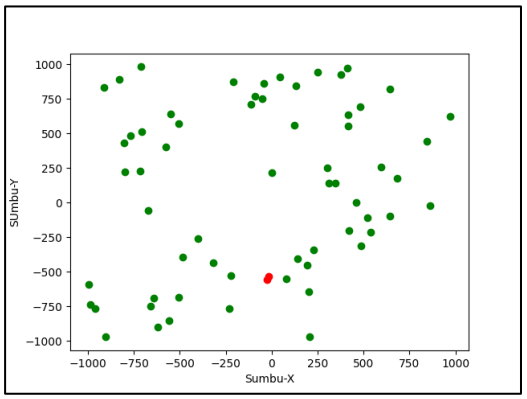
Gambar 3.2.3 Hasil Brute Force
Sumber: Dokumen Penulis



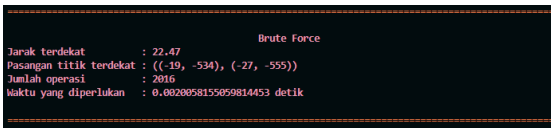
Gambar 3.2.4 Hasil Divide And Conquer
Sumber:Dokumen Penulis

- Ujicoba n = 64 pada 2 dimensi

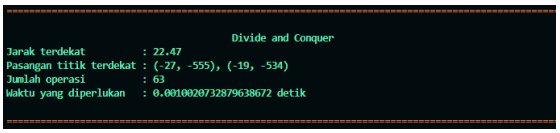
Gambar 3.3.1 Informasi Masukan
Sumber: Dokumen Penulis



Gambar 3.3.2 Visualisasi Titik-Titik
Sumber:Dokumen Penulis



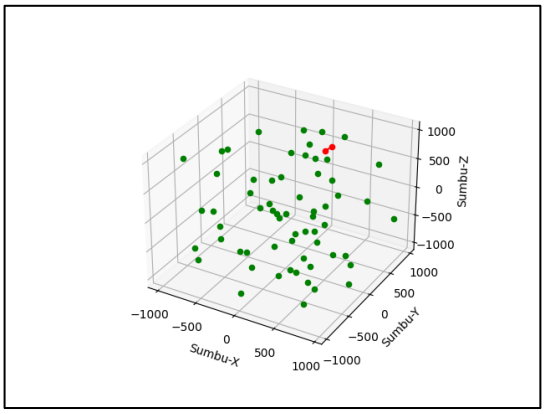
Gambar 3.3.3 Hasil Brute Force
Sumber: Dokumen Penulis



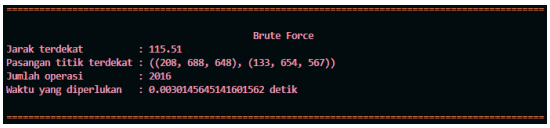
Gambar 3.3.4 Hasil Divide And Conquer
Sumber:Dokumen Penulis

- Ujicoba n = 64 pada 3 dimensi

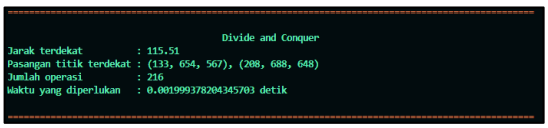
Gambar 3.4.1 Informasi Masukan
Sumber: Dokumen Penulis



Gambar 3.4.2 Visualisasi Titik-Titik
Sumber:Dokumen Penulis



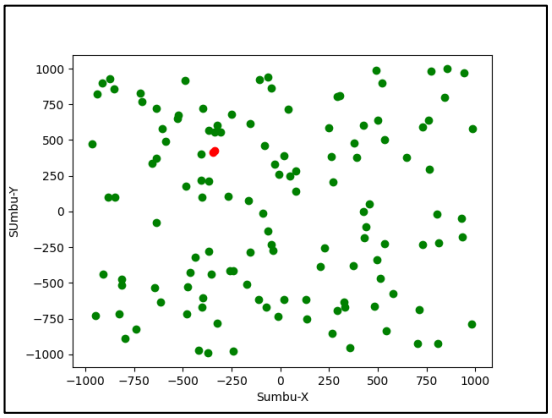
Gambar 3.4.3 Hasil Brute Force
Sumber: Dokumen Penulis



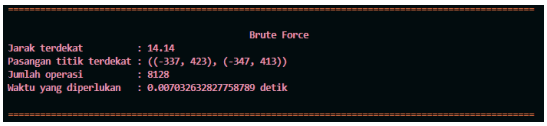
Gambar 3.4.4 Hasil Divide And Conquer
Sumber:Dokumen Penulis

- Ujicoba n = 128 pada 2 dimensi

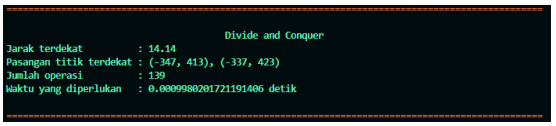
Gambar 3.5.1 Informasi Masukan
Sumber: Dokumen Penulis



Gambar 3.5.2 Visualisasi Titik-Titik
Sumber:Dokumen Penulis



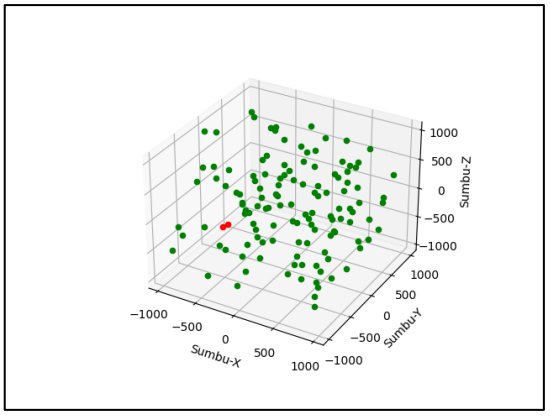
Gambar 3.5.3 Hasil Brute Force
Sumber: Dokumen Penulis



Gambar 3.5.4 Hasil Divide And Conquer
Sumber:Dokumen Penulis

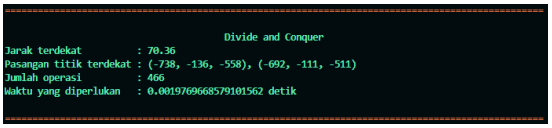
- Ujicoba n = 128 pada 3 dimensi

Gambar 3.6.1 Informasi Masukan
Sumber: Dokumen Penulis



Gambar 3.6.2 Visualisasi Titik-Titik
Sumber:Dokumen Penulis

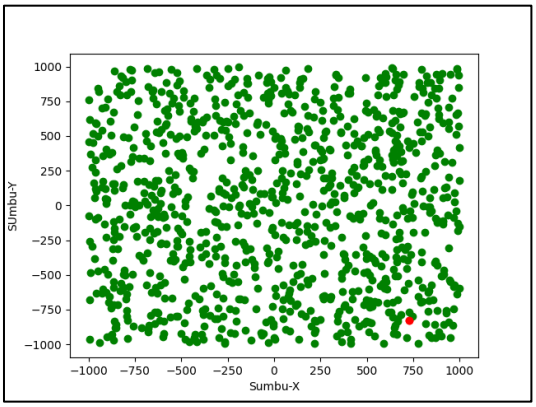
Gambar 3.6.3 Hasil Brute Force
Sumber: Dokumen Penulis



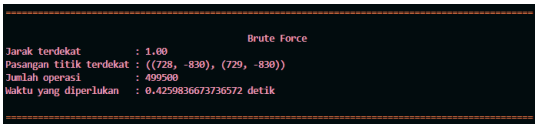
Gambar 3.6.4 Hasil Divide And Conquer
Sumber:Dokumen Penulis

- Ujicoba n = 1000 pada 2 dimensi

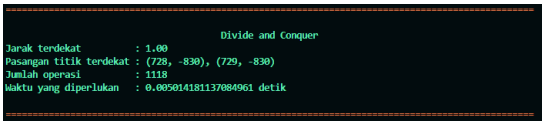
Gambar 3.7.1 Informasi Masukan
Sumber: Dokumen Penulis



Gambar 3.7.2 Visualisasi Titik-Titik
Sumber:Dokumen Penulis



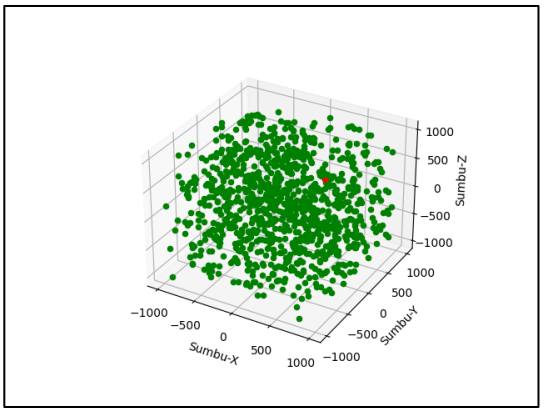
Gambar 3.7.3 Hasil Brute Force
Sumber: Dokumen Penulis



Gambar 3.7.4 Hasil Divide And Conquer
Sumber:Dokumen Penulis

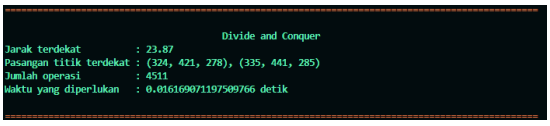
- Ujicoba n = 1000 pada 3 dimensi

Gambar 3.8.1 Informasi Masukan
Sumber: Dokumen Penulis



Gambar 3.8.2 Visualisasi Titik-Titik
Sumber:Dokumen Penulis

Gambar 3.8.3 Hasil Brute Force
Sumber: Dokumen Penulis



Gambar 3.8.4 Hasil Divide And Conquer
Sumber:Dokumen Penulis

- Ujicoba file input dan output

```
Tuple   :   1000
Dimensi :   3
1.  558, 132, 571
2.  524, 639, -332
3.  -295, -977, 765
4.  190, 384, 981
5.  282, 573, -484
6.  892, -411, -715
7.  -353, -155, -842
8.  227, 331, 938
9.  -964, -967, 239
10. 455, 930, -571
11. 908, -157, -97
12. -603, -771, -877
13. -134, 999, 486
14. 872, 616, -497
15. 639, -310, -390
16. 251, 285, -645
17. -692, -642, -36
18. -678, 887, 541
19. 130, -364, 750
20. -516, -763, 815
21. 337, 436, -58
22. 115, 893, -514
23. 665, 516, 126
24. -40, -136, -748
25. -923, -559, 882
26. 924, 887, -352
27. -437, 662, 202
28. -581, 55, 582
29. 481, -654, 987
30. -306, 425, 717
31. -290, -722, 382
32. -710, 182, 644
33. 68, 380, 191
34. -596, 560, 953
35. -687, 724, 535
36. -548, 385, -580
37. -372, 224, -963
38. 547, 648, -296
39. 60, 979, -112
40. -701, -88, 19
41. 332, 399, 561
42. -511, 171, -586
```

Gambar 3.9.1 Input.txt
Sumber: Dokumen Penulis

```
==========================================================================================
Jumlah tuple            : 1000
Dimensi vektor          : 3
Nilai minimum           : -1000
Nilai maksimum          : 1000
==========================================================================================
                                    Divide and Conquer
Jarak terdekat          : 23.87
Pasangan titik terdekat : (324, 421, 278), (335, 441, 285)
Jumlah operasi          : 4511
Waktu yang diperlukan   : 0.016169071197509766 detik
==========================================================================================
                                        Brute Force
Jarak terdekat          : 23.87
Pasangan titik terdekat : ((324, 421, 278), (335, 441, 285))
Jumlah operasi          : 499500
Waktu yang diperlukan   : 0.5794520378112793 detik
==========================================================================================
```

Gambar 3.9.2 Output.txt
Sumber:Dokumen Penulis

Bab 4
Kode Program dalam Bahasa *Python*

Berikut ini adalah kode program penulis yang dapat menyelesaikan permasalahan *closest pair* dengan metode *Brute Force* dan *Divide and Conquer* serta ditulis dalam bahasa pemrograman *python*.

4.1 main.py

Pada kode program ini akan dijalankan alur program secara terurut dengan memanggil kumpulan fungsi-fungsi perantara yang terdapat pada file calculation.py dan visualization.py.

```
from calculation import closestPairDNC, closestPairBruteForce,
inputRandom
from visualization import *
import time
import os


if __name__ == "__main__":
    PATH = os.path.dirname(os.path.realpath(__file__))
    mainLagi = True
    while(mainLagi == True):
        print("\033[1;31m" +
"=================================================================
================================" + "\033[0m")
        print("\033[1;35m" + "              .d88b 8    .d88b. .d88b.
8888 .d88b. 88888    888b.    db    888 888b. ")
        print("              8P    8    8P  Y8 YPwww. 8www
YPwww.   8      8  .8   dPYb    8  8  .8 ")
        print("              8b    8    8b  d8     d8
8        d8   8      8wwP'  dPwwYb   8  8wwK' ")
        print("              `Y88P 8888 `Y88P' `Y88P' 8888
`Y88P'   8      8     dP    Yb 888 8  Yb " + "\033[0m")
        print("\033[1;32m" +
"              P    R    O    B    L    E    M            S    O
 L    V    E    R               " + "\033[0m")
        print("\033[1;36m" + "                                      by :
M. Rifqi F. / 13521166              " + "\033[0m")
        print("\033[1;31m" +
"=================================================================
================================" + "\033[0m")

        print("\033[1;33m")
        count = int(input(">> Masukkan jumlah tuple: "))
        dimension = int(input(">> Masukkan dimensi vektor: "))
        print("\033[0m")

        while(count < 2 or dimension < 0):
            print("\033[1;31m" + "Jumlah tuple dan dimensi vektor
tidak valid!" + "\033[0m")
            print("\033[1;33m")
            count = int(input(">> Masukkan jumlah tuple: "))
            dimension = int(input(">> Masukkan dimensi vektor: "))
            print("\033[0m")
```

```
        print("\033[1;34m")
        minVal = int(input(">> Masukkan nilai minimum: "))
        maxVal = int(input(">> Masukkan nilai maksimum: "))
        print("\033[0m")

        while(minVal >= maxVal):
            print("\033[1;31m" + "Nilai minimum harus lebih kecil
dari nilai maksimum!" + "\033[0m")
            print("\033[1;34m")
            minVal = int(input(">> Masukkan nilai minimum: "))
            maxVal = int(input(">> Masukkan nilai maksimum: "))
            print("\033[0m")

        vectorList = inputRandom(count, dimension, minVal, maxVal)
        # print a message to inform the user that the generated
vectors have been saved to "input.txt"
        print("\033[1;36m" + "Point-point hasil input random telah
disimpan dalam file input.txt pada folder input" + "\033[0m")
        print("\033[1;31m" +
"=================================================================
================================" + "\033[0m")

        # DIVIDE AND CONQUER
        # waktu awal
        start_time = time.time()
        # perhitungan jarak terdekat
        counterDivideAndConquer = [0]
        d, res1, res2 = closestPairDNC(vectorList, count,
counterDivideAndConquer)
        # waktu selesai
        end_time = time.time()
        # selisih waktu
        total_time = end_time - start_time
        print("\033[1;32m")
        print("                                          Divide and
Conquer")
        print(f"Jarak terdekat\t\t: {d:.2f}")
        print(f"Pasangan titik terdekat\t: {res1}, {res2}")
        print(f"Jumlah operasi\t\t: {counterDivideAndConquer[0]}")
        print("Waktu yang diperlukan\t:", total_time, "detik")
        print("\033[0m")

        print("\033[1;31m" +
"=================================================================
================================" + "\033[0m")
        # BRUTE FORCE
        # waktu awal
        start_time = time.time()
        # perhitungan jarak terdekat
```

```
        counterBruteForce = int((len(vectorList)-1) *
len(vectorList) / 2)
        best_dist, res3, res4 = closestPairBruteForce(vectorList)
        # waktu selesai
        end_time = time.time()
        # selisih waktu
        time_total = end_time - start_time
        print("\033[1;35m")
        print("                                                Brute
Force ")
        print(f"Jarak terdekat\t\t: {best_dist:.2f}")
        print(f"Pasangan titik terdekat\t: {res3, res4}")
        print(f"Jumlah operasi\t\t: {counterBruteForce}")
        print("Waktu yang diperlukan\t:", time_total, "detik")
        print("\033[0m")

        # membuka file teks baru untuk menulis
        with open(f"{PATH}/output/output.txt", "w") as f:
            # menuliskan hasil keluaran ke dalam file
            f.write(f"==============================================
================================================\n")
            f.write(f"Jumlah tuple\t\t\t: {count}\n")
            f.write(f"Dimensi vektor\t\t\t: {dimension}\n")
            f.write(f"Nilai minimum\t\t\t: {minVal}\n")
            f.write(f"Nilai maksimum\t\t\t: {maxVal}\n")
            f.write(f"==============================================
================================================\n")
            f.write(f"                                          Divide
and Conquer\n")
            f.write(f"Jarak terdekat\t\t\t: {d:.2f}\n")
            f.write(f"Pasangan titik terdekat\t: {res1}, {res2}\n")
            f.write(f"Jumlah operasi\t\t\t:
{counterDivideAndConquer[0]}\n")
            f.write(f"Waktu yang diperlukan\t: {total_time}
detik\n")
            f.write(f"==============================================
================================================\n")
            f.write(f"                                                Br
ute Force \n")
            f.write(f"Jarak terdekat\t\t\t: {best_dist:.2f}\n")
            f.write(f"Pasangan titik terdekat\t: {res3, res4}\n")
            f.write(f"Jumlah operasi\t\t\t: {counterBruteForce}\n")
            f.write(f"Waktu yang diperlukan\t: {time_total}
detik\n")
            f.write(f"==============================================
================================================\n")

        print("\033[1;31m" +
"================================================================
==============================" + "\033[0m")
```

```
        if(dimension == 3 or dimension == 2):
            print("\033[1;33m")
            show = input(">> Apakah Anda ingin menampilkan grafik?
(y/n): ")
            print("\033[0m")
            while(show != "y" and show != "n"):
                print("Input tidak valid!")
                print("\033[1;33m")
                show = input(">> Apakah Anda ingin menampilkan
grafik? (y/n): ")
                print("\033[0m")
            if(show == "y"):
                if (dimension == 3):
                    show3d(vectorList, res1, res2)
                if (dimension == 2):
                    show2d(vectorList, res1, res2)
        print("\033[1;34m")
        main = input(">> Apakah Anda ingin main lagi? (y/n): ")
        print("\033[0m")
        while(main != "y" and main != "n"):
            print("\033[1;31m" + "Input tidak valid!" + "\033[0m")
            print("\033[1;34m")
            main = input(">> Apakah Anda ingin mulai lagi? (y/n): ")
            print("\033[0m")
        if(main == "n"):
            mainLagi = False
    print("\033[1;31m" +
"=================================================================
================================" + "\033[0m")
    print("\033[1;32m" +
"                                        Thank You :)" + "\033[0m")
    print("\033[1;31m" +
"=================================================================
================================" + "\033[0m")
```

4.2 calculation.py

Pada kode ini terdapat beberapa fungsi yang digunakan oleh program utama untuk melakukan perhitungan.

```
from typing import List
import math
import random
import os

def euclideanDistance(p1, p2):
    # Initialize sum to zero
    sum = 0
    # Iterate over each dimension of the points
    for i in range(len(p1)):
        # Calculate the difference between the corresponding
coordinates
        diff = p1[i] - p2[i]
```

```
        # Square the difference and add it to the sum
        sum += diff ** 2
    # Take the square root of the sum to get the Euclidean distance
    distance = math.sqrt(sum)
    return distance

def distanceOf3(P, counterDivideAndConquer):
    # Calculate the distances between each pair of points using the
Euclidean distance function
    d01 = euclideanDistance(P[0], P[1])
    counterDivideAndConquer[0] += 1
    d02 = euclideanDistance(P[0], P[2])
    counterDivideAndConquer[0] += 1
    d12 = euclideanDistance(P[1], P[2])
    counterDivideAndConquer[0] += 1
    # Compare the distances to find the minimum distance and the
corresponding points
    if (d01 < d02):
        if (d01 < d12):
            # Distance between P[0] and P[1] is smallest
            return d01, P[0], P[1]
        else:
            # Distance between P[1] and P[2] is smallest
            return d12, P[1], P[2]
    else:
        if (d02 < d12):
            # Distance between P[0] and P[2] is smallest
            return d02, P[0], P[2]
        else:
            # Distance between P[1] and P[2] is smallest
            return d12, P[1], P[2]

def closestPairDNC(P: List, n: int, counterDivideAndConquer):
# Base case when there are only two points
    # P = P.sort(key=lambda x: x[0])
    P = sorted(P, key=lambda x: x[0])
    # print(P)
    if n == 2:
        d = euclideanDistance(P[0], P[1])
        counterDivideAndConquer[0] += 1
        return d, P[0], P[1]
    if n == 3:
        return distanceOf3(P, counterDivideAndConquer)
    # Divide the set into two halves
    mid = n // 2
    S1 = P[:(mid+(n%2))]
    S2 = P[(mid+(n%2)):]
    # Recursive calls to find the closest pair in each half
    d1, p1, p2 = closestPairDNC(S1, mid+(n%2),
counterDivideAndConquer)
```

```
    d2, q1, q2 = closestPairDNC(S2, n-(mid+(n%2)),
counterDivideAndConquer)
    d, p1, p2 = (d1, p1, p2) if d1 < d2 else (d2, q1, q2)
    # Find the minimum distance between the two halves
    d = min(d1, d2)
    # Check for closest pair across the two halves
    # result = []
    closest_pair = (d, p1, p2)
    strip = []
    for i in range(n):
        if abs(P[i][0] - P[mid+(n%2)][0]) < d:
            strip.append(P[i])
    strip.sort(key=lambda x: x[1])
    size = len(strip)
    for i in range(size):
        for j in range(i+1, size):
            if strip[j][1] - strip[i][1] >= d:
                continue
            else:
                distance = euclideanDistance(strip[i],strip[j])
                counterDivideAndConquer[0] += 1
                d = min(d, distance)
                if (d == distance):
                    closest_pair = (d,strip[i],strip[j])
    return closest_pair

def closestPairBruteForce(P):
    # Calculate the length of the input list
    n = len(P)
    # Set initial values for the best distance and best pair
    best_dist = float('inf')
    best_pair = None
    # Loop through all pairs of points and calculate their distance
    for i in range(n):
        for j in range(i+1, n):
            dist = euclideanDistance(P[i], P[j])
            # If the distance between the pair of points is less
than the best distance,
            # update the best distance and best pair
            if dist < best_dist:
                best_dist = dist
                best_pair = (P[i], P[j])
    # Return the best distance and the two points that form the best
pair
    return best_dist, best_pair[0], best_pair[1]

def inputRandom(count, dimension, minVal, maxVal):
    PATH = os.path.dirname(os.path.realpath(__file__))
    vectorList = []
```

```
    # Generate a list of random vectors with the specified number of
dimensions
    for i in range(count):
        vector = ()
        for j in range(dimension):
            vector += (random.randint(minVal, maxVal),)
        vectorList.append(vector)
    # Write the list of vectors to a file named "input.txt"
    with open(f"{PATH}/input/input.txt", "w") as outFile:
        outFile.write("Tuple\t:\t" + str(count) + "\n" +
"Dimensi\t:\t" + str(dimension) + "\n")
        i = 1
        for vector in vectorList:
            outFile.write(str(i) + ".\t")
            for idx, component in enumerate(vector):
                if idx != len(vector) - 1:
                    outFile.write(str(component) + ", ")
                else:
                    outFile.write(str(component))
            outFile.write("\n")
            i += 1
    return vectorList
```

4.3 visualization.py

Pada kode ini terdapat beberapa fungsi yang digunakan oleh program utama untuk melakukan visualisasi dari titik-titik terkait.

```
import matplotlib.pyplot as plt


def show3d(vectorList, res1, res2):
    # create 3D object axes
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    # Adding points to the plot
    for point in vectorList:
        if point == res1 or point == res2:
            ax.scatter(point[0], point[1], point[2], c = "red") #
red marker used for closest points
        else:
            ax.scatter(point[0], point[1], point[2], c = "green") #
green marker used for other points
    # Adding line to the plot
    xs = [res1[0], res2[0]]
    ys = [res1[1], res2[1]]
    zs = [res1[2], res2[2]]
    ax.plot(xs, ys, zs, c="red") # red line to connect closest
points

    ax.set_xlabel('Sumbu-X') # Adding x-axis label
    ax.set_ylabel('Sumbu-Y') # Adding y-axis label
```

```
    ax.set_zlabel('Sumbu-Z') # Adding z-axis label

    plt.show()

def show2d(vectorList, res1, res2):
    # create 2D object axes
    fig = plt.figure()
    ax = fig.add_subplot(111)
    # Adding points to the plot
    for point in vectorList:
        if point == res1 or point == res2:
            ax.scatter(point[0], point[1], c = "red") # red marker
used for closest points
        else:
            ax.scatter(point[0], point[1], c = "green") # green
marker used for other points
    # Adding line to the plot
    ax.plot([res1[0], res2[0]], [res1[1], res2[1]], c="red") # red
line to connect closest points
        # Set x and y axis labels
    ax.set_xlabel('Sumbu-X') # Adding x-axis label
    ax.set_ylabel('SUmbu-Y') # Adding y-axis label
    plt.show()
```

Repository program ini dapat diakses melalui pranala berikut ini.

Bab 5
Tabel Penilaian

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

Bab 6
Kesimpulan

Dalam melakukan penyelesaian permasalahan *closest pair* dengan komputasi terdapat berbagai macam metode yang dapat diaplikasikan. Salah satunya adalah dengan algoritma *Divide and Conquer*, yaitu: dengan membagi kumpulan titik menjadi beberapa sub-kumpulan titik untuk dicari jarak terkecil untuk dijadikan minimum lokal, kemudian antar minimum lokal dibandingkan hingga diperoleh minimum global.

Pada Tugas Kecil 2 Strategi Algoritma ini, penulis membuat sebuah aplikasi berbasis *Command Line Interface* untuk menyelesaikan permasalahan *closest pair* yang ditulis dalam bahasa pemrograman python. Aplikasi penulis berhasil melakukan penyelesaian permasalahan dengan cukup baik, walaupun masih memiliki tingkat efektifitas dan efisiensi yang belum begitu baik.

Bab 7
Referensi

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2010-2011/Makalah2010/MakalahStima2010-055.pdf (diakses pada 28 Februari 2023)

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf (diakses pada 28 Februari 2023)

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf (diakses pada 28 Februari 2023)

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian3.pdf (diakses pada 28 Februari 2023)

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Divide-and-Conquer-(2022)-Bagian4.pdf (diakses pada 28 Februari 2023)