TUGAS KECIL 2 IF2211 STRATEGI ALGORITMA

SEMESTER 2 TAHUN 2022/2023

MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA DIVIDE AND CONQUER

Disusun Oleh:

Jericho Russel Sebastian

NIM 13521107

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

I. Latar Belakang

Pencarian pasangan titik terdekat merupakan masalah yang dirumuskan sebagai berikut: diberikan sebuah kumpulan titik dalam ruang $\mathbb{R}^n$; untuk semua pasangan titik yang mungkin dibentuk, carilah pasangan titik dengan jarak Euclidean terkecil. Masalah pencarian pasangan titik terdekat merupakan masalah yang timbul dalam berbagai aplikasi di dunia nyata, seperti pengenalan citra wajah, mesin pencari kata, pengenalan pola, dan sebagainya.

Terdapat berbagai pendekatan yang dapat digunakan dalam menyelesaikan masalah ini. Pendekatan yang paling sederhana dan lempang adalah dengan menggunakan teknik *brute force*, yaitu dengan memeriksa seluruh kemungkinan pasangan titik yang mungkin untuk menentukan pasangan titik dengan jarak terkecil. Namun, karena ketidaksangkilan algoritma *brute force*, masalah ini seringkali dipecahkan dengan strategi lain yang dapat memberikan hasil yang sama dalam waktu yang lebih cepat. Salah satu strategi yang dapat digunakan adalah *divide and conquer*.

Dengan menggunakan pendekatan *divide and conquer*, penulis menyusun WowBestMatch, sebuah program sederhana untuk memecahkan masalah pencarian pasangan titik terdekat.

II. Pembahasan

2.1 Algoritma *Divide and Conquer*

Divide and conquer merupakan strategi algoritma rekursif yang terdiri dari 3 tahap:

1. *Divide*; bagi masalah menjadi beberapa upa-masalah yang sejenis dengan masalah awal hingga upa-masalah cukup kecil.
2. *Conquer*; selesaikan upa-masalah secara langsung.
3. *Combine*; rangkai solusi dari upa-masalah untuk membentuk solusi dari masalah secara keseluruhan.

Berdasarkan metode pembagian dan penggabungan masalah yang digunakan, algoritma *divide and conquer* dapat dikelompokkan ke dalam:

1. *Easy-split*/*hard-join*; tahap *divide* sederhana, namun tahap *combine* rumit dan/atau memakan waktu.
2. *Hard-split*/*easy-join*; tahap *divide* rumit dan/atau memakan waktu, namun tahap *combine* sederhana.

Analisis kompleksitas algoritma *divide and conquer* tergolong rumit akibat rekursivitas yang mempersulit perhitungan. Jon Louis Bentley, Dorothea Blostein, dan James B. Saxe mengemukakan teorema master pada tahun 1980 untuk menganalisis kompleksitas waktu algoritma *divide and conquer* sebagai berikut: diberikan sebuah algoritma rekursif dengan fungsi waktu sebagai berikut:

$$T(n) = aT\left(\frac{n}{b}\right) + cn^d$$

Kompleksitas waktu asimtotik dari algoritma tersebut adalah:

$$T(n) = \begin{cases} O(n^d) & , a < b^d \\ O(n^d \log n) & , a = b^d \\ O\left(n^{\log_b a}\right) & , a > b^d \end{cases}$$

2.2 Jarak Euclidean

Jarak Euclidean antara dua titik dalam ruang $\mathbb{R}^n$ didefinisikan sebagai berikut: diberikan dua buah titik $A = a_1\,\hat{\imath}_1 + a_2\hat{\imath}_2 + \cdots + a_n\hat{\imath}_n$ dan $B = b_1\hat{\imath}_1 + b_2\hat{\imath}_2 + \cdots + b_n\hat{\imath}_n$, jarak Euclidean antara A dan B adalah d sedemikian hingga:

$$d = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + \cdots + (b_n - a_n)^2}$$

2.3 Algoritma Pencarian Pasangan Titik Terdekat

Algoritma pencarian pasangan titik terdekat yang digunakan oleh program WowBestMatch adalah algoritma *easy-split*/*hard-join* sebagai berikut: diberikan sebuah larik berisi n titik (dinyatakan dengan vektor posisi) dalam ruang $\mathbb{R}^k$ terurut menaik berdasarkan komponen pertama dengan $n \geq 1$, $k \geq 1$, algoritma akan melakukan tahap-tahap berikut:

1. Jika $n = 1$, tidak ada pasangan yang dapat dibentuk; hentikan pencarian.
2. Jika $n = 2$, pasangan titik terdekat adalah satu-satunya pasangan yang dapat dibentuk, yaitu dengan titik pertama dan kedua dalam larik; hentikan pencarian.
3. Jika $n \geq 3$, pilih sebuah nilai x sehingga $\frac{n}{2}$ titik pertama dalam larik memiliki komponen pertama $p_1 < x$ (dikelompokkan sebagai U) dan $\frac{n}{2}$ titik terakhir memiliki komponen pertama $p_1 > x$ (dikelompokkan sebagai V).
4. Ulangi algoritma ini untuk U dan V.
5. Anggap d_1 dan d_2 pasangan titik terdekat dalam U dan V. Anggap d pasangan dengan jarak terpendek antara d_1 dan d_2, atau pasangan yang ditemukan jika salah satu pencarian tidak menemukan pasangan titik.
6. Periksa pasangan-pasangan titik yang dapat dibentuk dari titik-titik dalam U dan titik-titik dalam V yang komponen pertamanya berjarak maksimum d dari x. Anggap d_3 pasangan titik dengan jarak minimum yang ditemukan. Pasangan titik terdekat adalah pasangan dengan jarak terpendek antara d dan d_3; hentikan pencarian.

III. Implementasi

WowBestMatch terdiri dari dua program, yaitu:

1. WowBM; program pencarian pasangan titik terdekat yang ditulis dalam bahasa C++.
2. WowPlot; program visualisasi yang ditulis dalam bahasa Python.

Sistematika file WowBestMatch adalah sebagai berikut:

```
Tucil2_13521107
├-- bin
|   ├-- wowbm.exe
|   ├-- wowbm_debug.exe
|   └-- wowplot.exe
├-- doc
|   └-- Tucil2_K1_13521107_Jericho Russel Sebastian.pdf
├-- src
|   ├-- wowbm
|   |   ├-- struct
|   |   |   ├-- exc
|   |   |   |   ├-- Exception.cpp
|   |   |   |   └-- Exception.hpp
|   |   |   └-- vec
|   |   |       ├-- Vector.cpp
|   |   |       ├-- Vector.hpp
|   |   |       ├-- VectorList.cpp
|   |   |       ├-- VectorList.hpp
|   |   |       ├-- VectorPair.cpp
|   |   |       └-- VectorPair.hpp
|   |   ├-- util
|   |   |   ├-- Random.cpp
|   |   |   ├-- Random.hpp
|   |   |   ├-- Timer.cpp
|   |   |   └-- Timer.hpp
|   |   └-- wowbm.cpp
|   └-- wowplot
|       ├-- __init__.py
|       └-- wowplot.py
├-- test
```

```
|   └── data1.csv
├── debug.bat
├── debug.sh
├── run.bat
├── run.sh
├── makefile
└── README.md
```

Penulis tidak akan menampilkan keseluruhan isi *source code* program, melainkan hanya deklarasi struktur kelas dan definisi metode yang relevan dengan pemecahan masalah pencarian serta penerapan algoritma *divide and conquer*.

3.1 Vector

Kelas `Vector` dideklarasikan pada file Vector.hpp. Sebuah objek `Vector` mewakili sebuah vektor posisi dari suatu titik dalam $\mathbb{R}^k$ dan memuat atribut berupa nilai k (banyak komponen) dan sebuah larik berisi nilai masing-masing komponen. Atribut *static* `distComputeCount` berguna untuk menghitung banyak pemanggilan fungsi `distance()`.

```
class Vector{
    public:
        Vector(int dimension);
        Vector(const Vector& other);
        ~Vector();

        void setComponent(int axis, double value);
        int getDimension() const;

        Vector& operator=(const Vector& other);
        double& operator[](int axis) const;
        int compare(const Vector& other) const;


        static int computeCount();
        static void resetComputeCount();
        static double distanceSqr(const Vector& v1, const Vector&
v2);
        static double distance(const Vector& v1, const Vector& v2);
        ...
    private:
        static int distComputeCount;
```

```
        int dimension;
        double* components;
};
```

3.2 Vector Pair

Kelas `VectorPair` dideklarasikan pada file VectorPair.hpp. Sebuah objek `VectorPair` memuat informasi mengenai dua buah `Vector` beserta jarak Euclidean antara keduanya.

```
class VectorPair{
    public:
        VectorPair(Vector& v1, Vector& v2, int i, int j);
        VectorPair(Vector* v1, Vector* v2, int i, int j);

        Vector V1() const;
        Vector V2() const;
        int I() const;
        int J() const;
        double distance() const;

        bool operator<(const VectorPair& other) const;
        ...
    private:
        Vector* v1;
        Vector* v2;
        int i;
        int j;
        double dist;
};
```

3.3 Vector List

Kelas `VectorList` dideklarasikan dalam file VectorList.hpp. Objek `VectorList` merupakan wadah untuk kumpulan titik yang akan diproses oleh program. Fungsi pencarian pasangan titik terdekat diimplementasi pada kelas ini.

```
class VectorList{
    public:
        static VectorList* randomOfSize(int size, int dimension);
        static VectorList* fromFile(const char* path);
```

```
        VectorList(int size, int dimension);
        ~VectorList();

        int getSize() const;

        void sort();
        VectorPair* closestPairBf();
        VectorPair* closestPairDnc();

        double loDistance(int i, int j) const;
        double loDistance(int i, double d) const;
        double maxDistance(int i, int j) const;
        Vector* operator[](int idx);
        const Vector operator[](int idx) const;
        ...
    private:
        int size;
        int dimension;
        Vector** buffer;

        void sort(int i, int j);
        int partition(int i, int j);
        VectorPair* closestPairDnc(int i, int j);
};
```

Fungsi `sort()` dan `partition()` merupakan implementasi algoritma *quicksort*, fungsi `closestPairBf()` merupakan implementasi algoritma pencarian pasangan titik terdekat secara *brute force*, dan fungsi `closestPairDnc()` merupakan implementasi algoritma pencarian pasangan titik terdekat secara *divide and conquer* sebagai berikut:

```
VectorPair* VectorList::closestPairDnc(int i, int j){
    if(j - i < 1){
        // Kasus n == 1, tidak ada pasangan
        return NULL;
    }else if(j - i == 1){
        // Kasus n == 2, satu pasangan
        return new VectorPair(buffer[i], buffer[j], i, j);
    }else{
        // Kasus n >= 3, bagi dua larik
        int k = (i+j)/2;
        VectorPair
            *d1 = closestPairDnc(i, k),
```

```
            *d2 = closestPairDnc(k+1, j),
            *d;

        // d adalah jarak terkecil antara d1 dan d2
        if(d1 && !d2){
            d = d1;
            delete d2;
        }else if(!d1 && d2){
            d = d2;
            delete d1;
        }else{
            if(*d1 < *d2){
                d = d1;
                delete d2;
            }else{
                d = d2;
                delete d1;
            }
        }

        // Tentukan batas titik-titik yang komponen pertamanya
berjarak d dari x
        double mid = (*buffer[k])[0] + loDistance(k, k+1)/2;
        int
            p = i,
            q = j;

        while(p < k && loDistance(p, mid) > d->distance())++p;
        while(q > k+1 && loDistance(q, mid) > d->distance())--q;

        for(int u = p; u <= k; ++u){
            for(int v = q; v >= k+1; --v){
                // Buang pasangan titik yang jarak salah satu
komponennya lebih dari d
                if(dimension > 1 && maxDistance(u, v) > d-
>distance())continue;
                VectorPair* d3 = new VectorPair(buffer[u],
buffer[v], u, v);

                // Cek apakah jarak d3 lebih pendek dari d
                if(*d3 < *d){
                    delete d;
                    d = d3;
                }else{
                    delete d3;
```

```
                }
            }
        }

        // Pasangan titik terdekat ditemukan
        return d;
    }
}
```

Algoritma di atas memiliki fungsi waktu sebagai berikut:

$$T(n) = \begin{cases} k & ,n = 2 \\ 2T\left(\frac{n}{2}\right) + cn & ,n \geq 3 \end{cases}$$

Menggunakan teorema master dengan $a = 2$, $b = 2$, $d = 1$, dapat dilihat bahwa $T(n) = O(n \log n)$. Hal ini menunjukkan bahwa algoritma ini lebih baik dibandingkan dengan algoritma *brute force* dengan kompleksitas waktu $O(n^2)$.

3.4 WowPlot

WowPlot, dalam file wowplot.py, merupakan program tambahan dalam bahasa Python yang digunakan untuk menampilkan visualisasi kumpulan titik sebagai sebuah *scatter plot* tiga dimensi. WowPlot membaca masukan dari file result.txt hasil luaran program WowBM dan memberikan informasi tambahan jika data lebih dari tiga dimensi, yaitu memberikan variasi ukuran titik dan warna untuk dimensi keempat dan kelima.

```
def sizeintp(t: float) -> float: ...
def colorintp(t: float) -> list[float]: ...
def slice_col(col: int, scl: float = 1) -> list[float]: ...
def sizes() -> list[float]: ...
def colors() -> list[list[float]]: ...

if __name__ == '__main__':
    Z = []
    try:
        # Baca masukan dari file result.txt
        file = open('result.txt', 'r')
        head = False
        for line in file:
            if not head:
```

```
            P = [int(d) for d in line.split(' ')]
            head = True
        else:
            Z.append([float(r) for r in line.split(' ')])
    file.close()
except OSError:
    print("Error while trying to open file ./result.txt")

# Gambar data dalam scatter plot 3D
l = len(Z)
if l > 0:
    ...

    fig = plot.figure()
    ax = fig.add_subplot(projection='3d')

    ...

    ax.scatter(
        slice_col(0),
        slice_col(1),
        slice_col(2),
        s=sizes(),
        c=colors()
    )

    ax.set_xlabel('Component 1')
    ax.set_ylabel('Component 2')
    ax.set_zlabel('Component 3')

    plot.show()
```

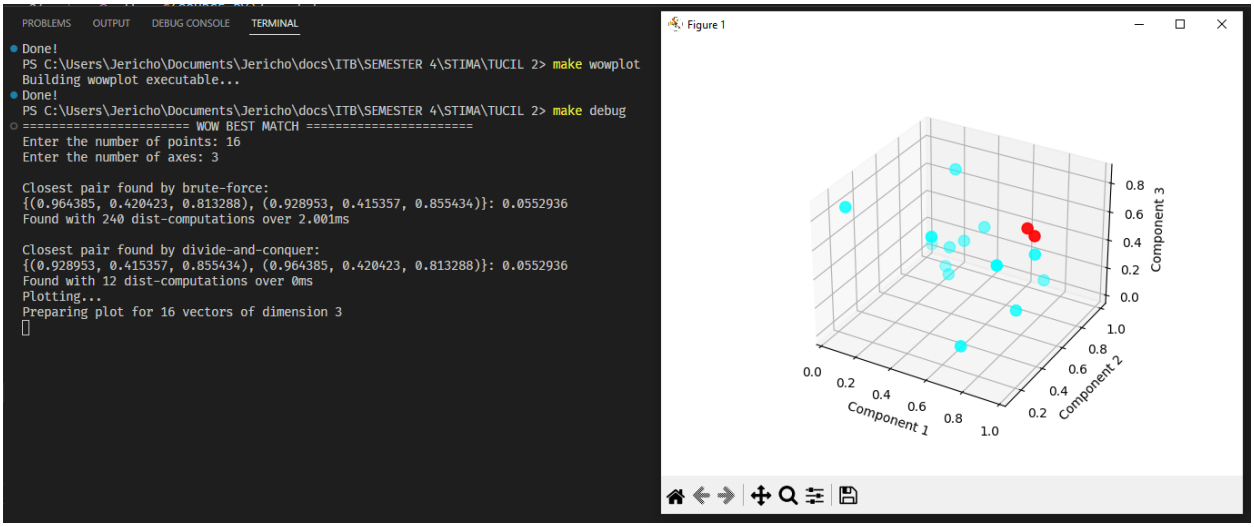
IV. Hasil Eksekusi

Berikut merupakan hasil eksekusi program WowBestMatch untuk beberapa contoh kasus yang diberikan, yaitu untuk nilai $n = 16$, $n = 64$, $n = 128$, dan $n = 1000$. Penulis akan menunjukkan variasi untuk nilai $k = 3$ dan $k = 5$. Perlu diperhatikan bahwa WowBM dapat menerima sembarang n, k dengan syarat $n \geq 0$, $k \geq 1$, namun hanya lima komponen pertama yang akan ditampilkan pada *scatter plot* oleh WowPlot.

Eksekusi program dilakukan di perangkat pribadi dengan spesifikasi sebagai berikut:

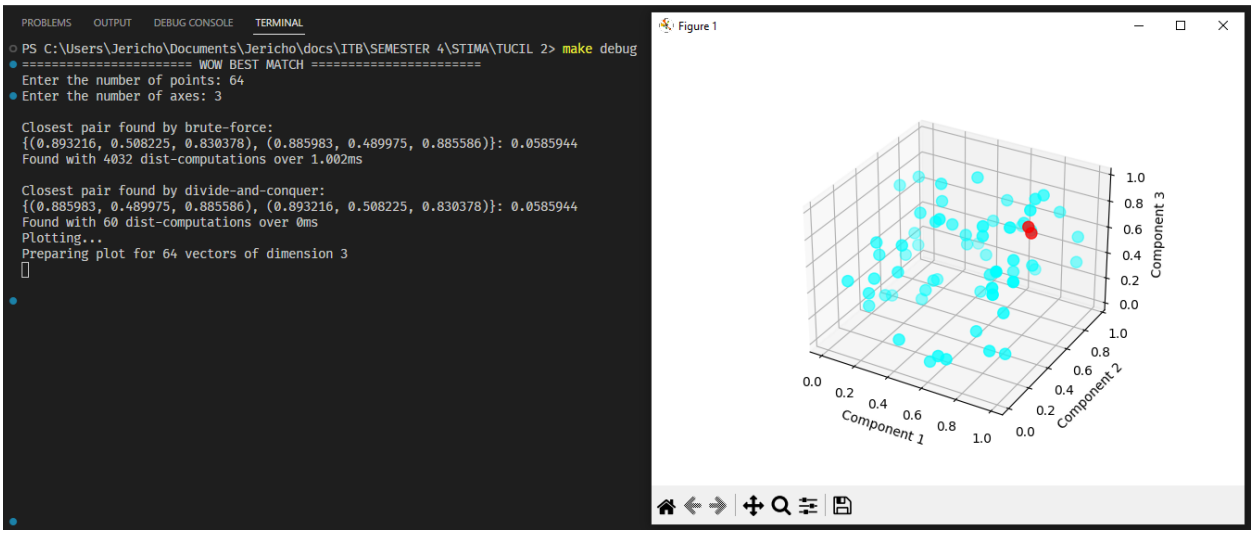
- Processor : Intel® Core™ i7-10510U ~2.3GHz 4 CPUs
- Installed RAM : 8.00 GB
- Operating System : Windows 10 Home 64-bit (10.0, Build 19044)
- System Type : x64-based PC

4.1 Kasus $n = 16, k = 3$



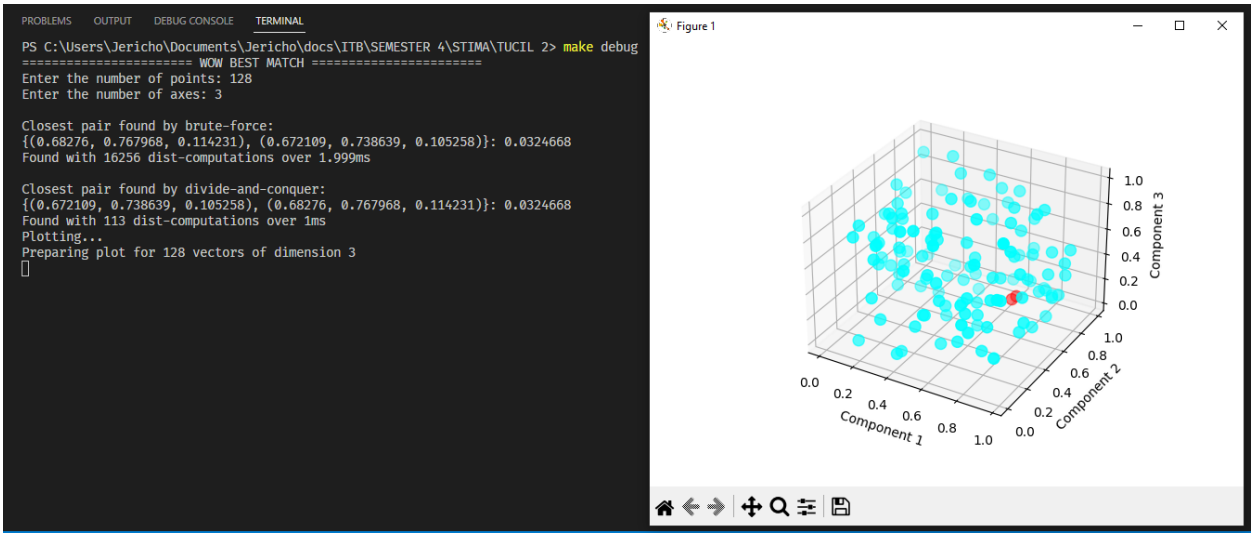
Gambar 4.1.1 Luaran WowBestMatch Untuk Kasus $n = 16$, $k = 3$

4.2 Kasus $n = 64, k = 3$



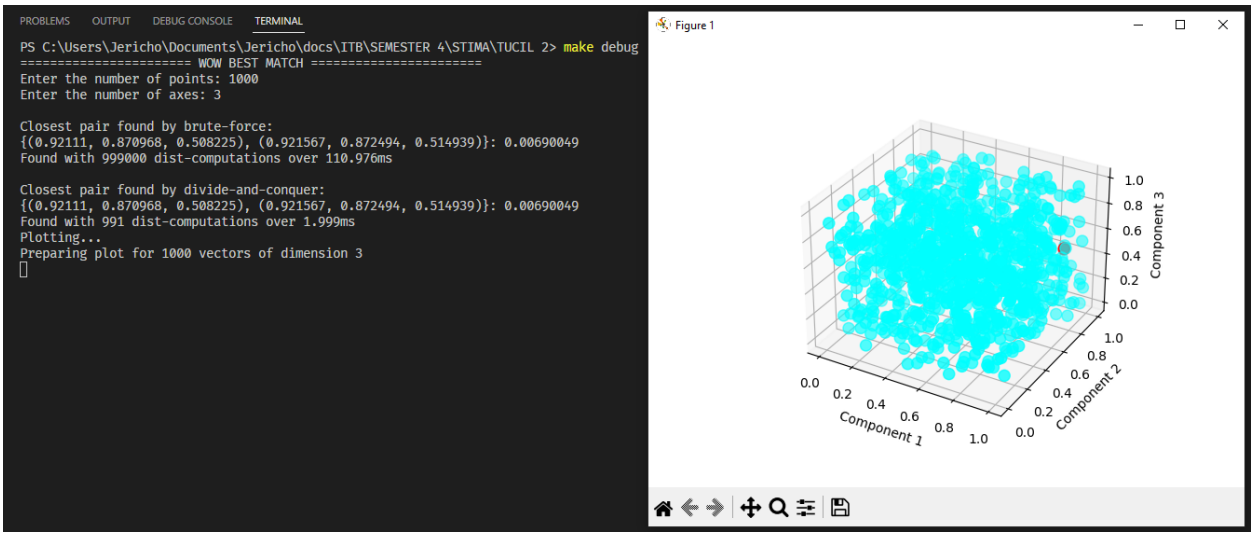
Gambar 4.2.1 Luaran WowBestMatch Untuk Kasus $n = 64$, $k = 3$

4.3 Kasus $n = 128, k = 3$



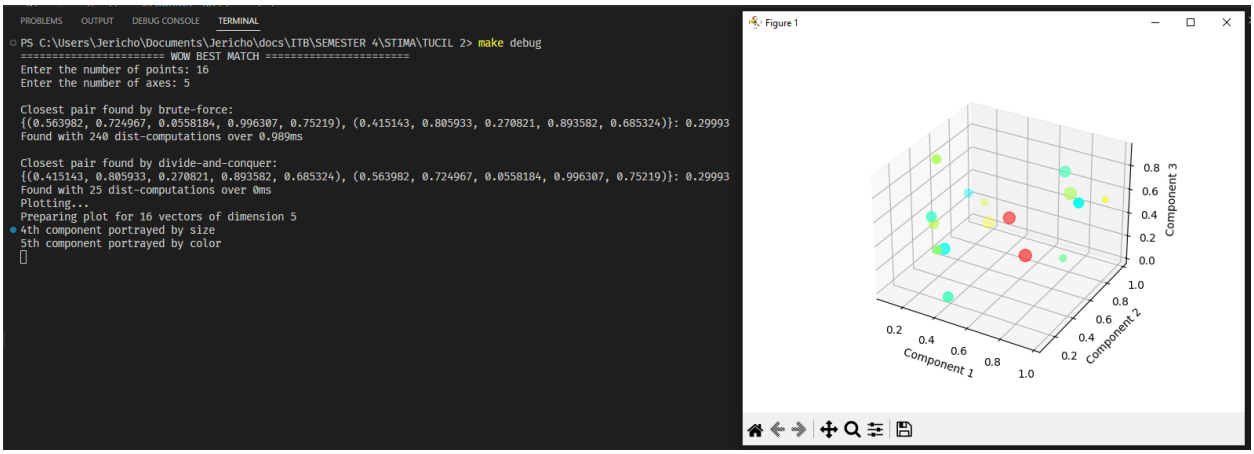
Gambar 4.3.1 Luaran WowBestMatch Untuk Kasus $n = 128$, $k = 3$

4.4 Kasus $n = 1000, k = 3$



Gambar 4.4.1 Luaran WowBestMatch Untuk Kasus $n = 1000, k = 3$

4.5 Kasus $n = 16, k = 5$

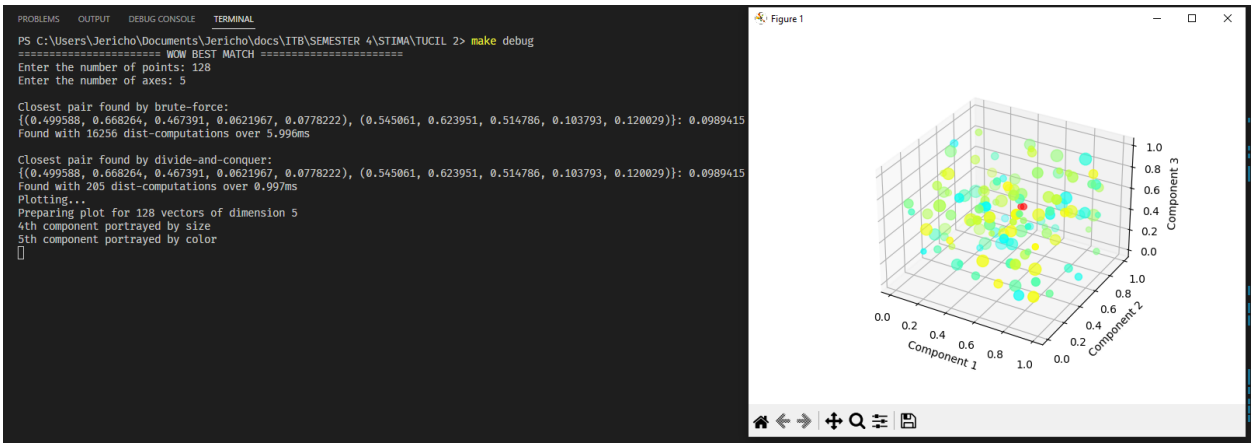


Gambar 4.5.1 Luaran WowBestMatch Untuk Kasus $n = 16, k = 5$

4.6 Kasus $n = 64, k = 5$

Gambar 4.6.1 Luaran WowBestMatch Untuk Kasus $n = 64$, $k = 5$

4.7 Kasus $n = 128, k = 5$



Gambar 4.7.1 Luaran WowBestMatch Untuk Kasus $n = 128$, $k = 5$

4.8 Kasus $n = 1000, k = 5$

Gambar 4.8.1 Luaran WowBestMatch Untuk Kasus $n = 1000, k = 5$

V. Lampiran

5.1 Repository GitHub

Repository untuk *source code* dan hasil kompilasi program ini dapat diakses lewat tautan berikut: https://github.com/JerichoFletcher/Tucil2_13521107

5.2 Tabel Penilaian

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa ada 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		

VI. Daftar Pustaka

Munir, Rinaldi. 2022. “Algoritma Divide and Conquer (Bagian 2)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf diakses pada 1 Maret 2023.

Bentley, Jon Louis; Haken, Dorothea; Saxe, James B.. 1980. “A General Method for Solving Divide-and-Conquer Recurrences”. ACM SIGACT News.