LAPORAN TUGAS KECIL 2 IF2211

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

Ditujukan untuk memenuhi tugas kecil 2 mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2022/2023

Disusun oleh:

Muhammad Abdul Aziz Ghazali	13521128
Muhammad Zaki Amanullah	13521146

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

2022

Daftar Isi

Contents

BAB I DESKRIPSI MASALAH

Mencari sepasang titik terdekat dengan Algoritma *Divide and Conquer* dapat dirumuskan untuk titik pada bidang datar (2D) dan bidang ruang (3D). Penulis mengembangkan algoritma mencari sepasang titik terdekat pada bidang 3D. Misalkan terdapat *n* buah titik pada ruang 3D. Setiap titik *P* di dalam ruang dinyatakan dengan koordinat $P = (x, y, z)$. Program akan mencari sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titk $P1 = (x1, y1, z1)$ dan $P2 = (x2, y2, z2)$ dihitung dengan rumus Euclidean berikut :

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

Penulis membuat program yang mencari pasangan titik dengan jarak *euclidean* terdekat dengan bahasa Python dengan menerapkan algoritma *divide and conquer* untuk penyelesaiannya, dan perbandingannya dengan algoritma *brute force*. Visualisasi dilakukan dengan library python matplotlib

BAB II TEORI SINGKAT

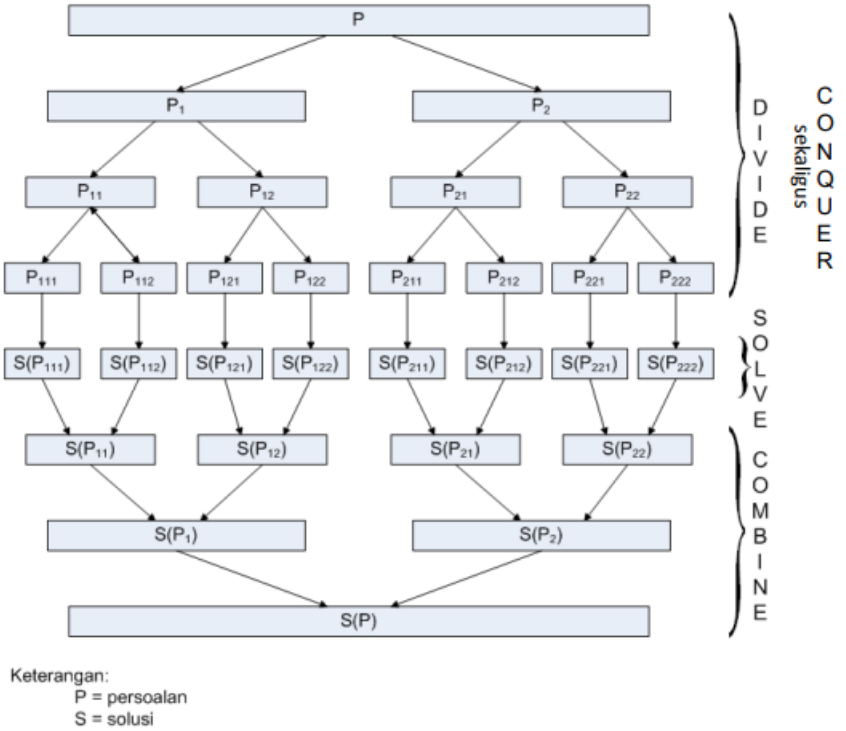
2.1 Algoritma *Divide and Conquer*

Dalam dunia ilmu komputer, dikenal suatu strategi fundamental yang banyak digunakan dalam banyak hal. Strategi *divide and conquer* langkahnya terbagi menjadi 3 , yaitu :

- Divide:
  membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya berukuran hampir sama),

- Conquer (solve):
  menyelesaikan masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar).

- Combine:
  mengabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.



Gambar 2.1.1 Ilustrasi Langkah Skema Umum *Divide and Conquer*

Algoritma ini secara umum memiliki kompleksitas berikut :

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

• T(n) : kompleksitas waktu penyelesaian persoalan P yang berukuran n

• g(n) : kompleksitas waktu untuk SOLVE jika n sudah berukuran kecil

• $T\ (n_1\ )+ T\ (n_2\ )\ldots + T\ (n_r)$: kompleksitas waktu untuk memproses setiap upa-persoalan

• f(n) : kompleksitas waktu untuk COMBINE solusi dari masing-masing upa-persoalan

Lalu, apabila pembagian selalu menghasilkan dua upa-persoalan yang berukuran sama, maka kompleksitasnya :

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

2.2 Jarak Euclidean

Euclidean distance adalah perhitungan jarak dari 2 buah titik dalam Euclidean space. Euclidean space diperkenalkan oleh Euclid, seorang matematikawan dari Yunani sekitar tahun 300 B.C.E. untuk mempelajari hubungan antara sudut dan jarak. Euclidean ini berkaitan dengan Teorema Phytagoras dan biasanya diterapkan pada 1, 2 dan 3 dimensi. Tapi juga sederhana jika diterapkan pada dimensi yang lebih tinggi.

Dalam dimensi ke-n, secara umum untuk dua titik yang diketahui koordinat kartesiannya, jarak kartesiannya adalah

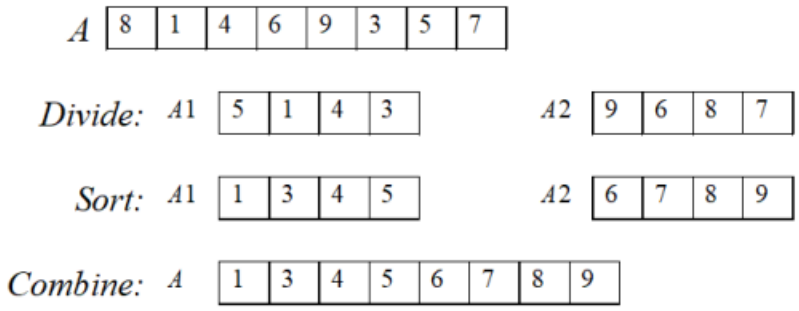
$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 \ldots + (p_n - q_n)^2}$$

Dengan p dan q masing masing adalah dua titik dalam dimensi-n yang sama

2.3 *Quicksort*

Quicksort adalah suatu algoritma pengurutan yang tercepat dengan pendekatan *divide and conquer*. Secara umum, *quicksort* akan membagi suatu array A menjadi 2 buah upa-array sebagaimana hingga :

Semua elemen di A1 ≤ Semua elemen di A2



Gambar 2.3.1 Ilustrasi Pembagian Array/Larik

Ada beberapa varian dari implementasi *quicksort*, yaitu :

- Varian Pertama / Versi Hoare
  (i) pilih x ∈ { A[1], A[2], ..., A[n] } sebagai pivot,
  (ii) pindai larik dari kiri sampai ditemukan elemen A[p] ≥ x
  (iii) pindai larik dari kanan sampai ditemukan elemen A[q] ≤ x
  (iv) pertukarkan A[p] ⟺ A[q]
  (v) ulangi (ii), dari posisi p + 1, dan (iii), dari posisi q – 1 , sampai kedua pemindaian bertemu di tengah larik (p ≥ q)

- Varian Kedua
  (i) Pilih x = A[0] atau elemen pertama pada partisi sebagai pivot
  (ii) Pindai dari elemen partisi dari kiri hingga menemukan yang lebih dari pivot
  (iii) Pindai elemen partisi dari kanan hingga menemukan yang kurang dari pivot
  (iv) Tukar kedua elemen dari langkah (ii) dan (iii)
  (v) Lakukan kembali dari langkah (ii) dari posisi pemindai terakhir hingga semua elemen telah dipindai.
  (vi) Pertukarkan pivot dengan elemen yang terakhir dipindai dari kiri
  (vii) Larik dipartisi menjadi 2, Elemen elemen di sebelah kiri dan kanan pivot.
  (viii) Untuk setiap partisi, lakukan dari langkah ke (i) hingga elemen partisi berjumlah 1

2.4 Algoritma Pencarian Pasangan Titik Terdekat

Berikut adalah langkah algoritma yang digunakan penulis dalam mencari pasangan titik dengan jarak titik terdekat dengan *divide and conquer*:

1. Larik yang berisi titik-titik akan dilakukan *sorting* atau pengurutan. Pengurutan ini berdasarkan nilai komponen absis pada titik. Program yang penulis buat melakukan *quicksort.*
2. Kemudian, larik akan dibagi menjadi 2 upalarik dengan nilai yang berada pada indeks larik paling tengah sebagai pivot. Lalu setiap upalarik akan mengulangi dari langkah 1 hingga hanya tersisa ≤3 titik pada larik.

3. Apabila telah tersisa ≤ 3 titik, maka akan dilakukan *brute force* pada larik titik tersebut untuk menghitung jarak pada setiap pasangan titik yang mungkin lalu mencari nilai minimum dari jarak sepasang titiknya.
4. Kemudian nilai jarak salahsatu komponen terdekat yang ditemukan pada larik tersebut akan dibandingkan dengan nilai jarak salahsatu komponen terdekat dengan larik pasangannya yang menjadi pasangan partisinya. Lalu didapatkan nilai jarak terdekat dari sebuah larik yang tadinya dipartisi menjadi 2 upalarik.
5. Lalu akan dicari larik *slab* yang akan berisi titik yang memiliki selisih jarak pada komponen absis dengan nilai komponen absis pivot lebih kecil daripada nilai yang didapat pada langkah sebelumnya pada masing masing partisi kedua upa larik.
6. Kemudian, akan dilakukan akan dicek pada setiap elemen kedua *slab* pada masing-masing upa larik dimana apabila ada titik pada *slab* pertama dan *slab* kedua yang selisih jaraknya kurang atau samadengan dari nilai yang didapat dari langkah ke 4, maka akan dilakukan pengecekan dimana apabila ternyata jarak dari kedua pasang titik dari masing-masing *slab* lebih pendek dari jarak terdekat yang didapat dari langkah ke-4 maka jarak terdekat pada pemartisian larik ini adalah jarak dari kedua elemen *slab* tersebut dan apabila nilainya sama dengan jarak yang didapat pada langkah ke-4, hanya akan dicatat elemen kedua *slab* tersebut juga merupakan pasangan dengan jarak titik terdekat. Proses eliminasi ini lebih efektif dibanding harus mengecek jarak setiap titik pada partisinya
7. Kemudian akan diulangi dari langkah ke-4 pada larik yang berada di tingkat upa larik proses ini hingga semua upa-larik telah disatukan (*combine and conquer)*
8. Pada akhirnya, kedua upa larik dari larik utama akan menghasilkan list pasangan titik terdekat dari larik keseluruhan.

BAB III IMPLEMENTASI PUSTAKA

3.1 Fungsi dan Prosedur

No.	Nama	Deskripsi	Parameter	Hasil
1	findDistance	Mencari nilai jarak antara 2 titik dalam dimnsi ke-n	a: titik pertama b: titik kedua	Bilangan rasional nilai jarak 2 titik
2	findClosestPair	Mencari pasangan titik dengan jarak terdekat menggunakan *divide and conquer* serta *brute force* secara rekursif	points: list dari titik dalam tuple dari titik koordinat kartesian n: banyaknya titik dalam points dimension: dimensi titik	List yang berisikan tuple berisikan titik-titik yang berjarak paling dekat.
3	findClosestPairOfBF	Mencari jarak terdekat dari sebuah list of points dengan menggunakan algoritma brute force	points: list dari titik-titik yang ingin dicari koordinat kartesian titik-titik yang paling dekat serta jaraknya.	List yang berisikan ttuple berisikan tiga buah nilai, dua nilai pertama merupakan pasangan titik-titik terdekat dan nilai ketiga adalah nilai jarak antara kedua titik.
4	partition	Sebuah prosedur yang akan melakukan swapping value pada sebuah list serta mengembalikan indeks tengah untuk partisi	arr: list of points low: index bawah high: index atas tuple_att_to_sort_to: index atribut dari tuple yang ingin di sorting	Array yang akan dimasukkan ke dalam parameter akan diubah dengan nilai pivot di tengah serta nilai-nilai yang lebih kecil di sebelah kiri pivot sedangkan nilai-nilai lebih besar di sebelah kanan pivot. Mengembalikan sebuah nilai yang merupakan nilai tengah partisi.
5	quickSort	Sebuah prosedur yang akan mengurutkan list of points dengan algoritma divide and	arr: list of points low: index bawah high: index atas tuple_att_to_sort_to:	Array yang dimasukkan ke dalam parameter akan terurut

		conquer quick sort untuk menghasilkan array yang terurut menaik berdasarkan atribut dari tuple terentu	index atribut dari tuple yang ingin di sorting	menaik berdasarkan atribut dari tuple
6	display	Sebuah prosedur yang akan menampilkan titik-titik pada bidang tiga dimensi dengan pasangan titik-titik terdekat berwarna merah dan titik-titik lain berwarna hijau. Prosedur ini akan berjalan jika dimensi dari list of points adalah tiga.	points: list of points closest_pair: tuple yang berisikan tiga atribut, dua atribut pertama merupakan pasangan titik-titik dengan jarak terdekat sedangkan atribut ketiga adalah jarak antara titik tersebut. dimension: jumlah dimensi dari list of points yang dimasukkan	Sebuah graf dalam bidang tiga dimensi yang berisikan titik-titik pada list of points yang akan berwarna merah apabila titik tersebut merupakan salah satu dari pasangan list terdekat dan akan berwarna hijau apabila bukan.

3.2 Implementasi

3.2.1 findDistance

```
def findDistance(a, b):
    global timesEuclideanDistanceCalculated
    res = 0
    for i in range(len(a)):
        res += (a[i] - b[i]) ** 2
    timesEuclideanDistanceCalculated += 1
    return math.sqrt(res)
```

3.2.2 findClosestPair

```python
def findClosestPair(points, n, dimension):
    if (n <= 3):
        return findClosestPairOfBF(points)
    else:
        # Sort the points on their abses
        quickSort(points, 0, len(points) - 1, 0)

        # Split the index into two arbitrary area, s1 and s2 and find the shortest pair in those two areas
        mid = n // 2
        s1 = points[:mid]
        s2 = points[mid:]
        shortest_s1 = findClosestPair(s1, len(s1), dimension)
        shortest_s2 = findClosestPair(s2, len(s2), dimension)
        if (shortest_s1[2] >= shortest_s2[2]) :
            shortest_s1_s2 = shortest_s2
        else :
            shortest_s1_s2 = shortest_s1

        # Find points in 'slab' such that those points have distance shorter than or equal to shortest_s1_s2 to the point in the mid index
        slab_s1 = [p for p in s1 if abs(p[0] - points[mid][0]) < shortest_s1_s2[2]]
        quickSort(slab_s1, 0, len(slab_s1) - 1, 1)
        slab_s2 = [p for p in s2 if abs(p[0] - points[mid][0]) < shortest_s1_s2[2]]
        quickSort(slab_s2, 0, len(slab_s2) - 1, 1)

        for i in range(len(slab_s1)):
            for j in range(len(slab_s2)):
                check = True
                for k in range(dimension):
                    if abs(slab_s1[i][k] - slab_s2[j][k]) > shortest_s1_s2[2]:
                        check = False
                if (check) :
                    tempDistance = findDistance(slab_s1[i], slab_s2[j])
                    if tempDistance < shortest_s1_s2[2]:
                        shortest_s1_s2 = (slab_s1[i], slab_s2[j], tempDistance)

        return shortest_s1_s2
```

3.2.3 FindClosestPairOfBF

```python
def findClosestPairOfBF(points):
    closest = ()
    min = float('inf')
    for i in range(len(points)):
        for j in range(i+1, len(points)):
            tempDistance = findDistance(points[i], points[j])
            if tempDistance < min:
                min = tempDistance
                closest = (points[i], points[j])
    return closest + (min,)
```

3.2.4 quickSort

```python
def quickSort(arr, low, high, tuple_att_to_sort):
    if (low < high):
        pi = partition(arr, low, high, tuple_att_to_sort)
        quickSort(arr, low, pi - 1, tuple_att_to_sort)
        quickSort(arr, pi + 1, high, tuple_att_to_sort)
```

3.2.5 partition

```python
def partition(arr, low, high, tuple_att_to_sort):
    i = low - 1
    pivot = arr[high][tuple_att_to_sort]
    for j in range(low, high):
        if (arr[j][tuple_att_to_sort] < pivot):
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1
```

3.2.6 display

```python
def display(points, closest_pair, dimension):
    if (dimension == 3):
        numpied_points = np.array(points)

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

        for x, y, z in points:
            if ((x, y, z) in closest_pair[:2]) :
                ax.scatter(x, y, z, marker='^', color='r')
            else:
                ax.scatter(x, y, z, marker='o', color='g')

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

        plt.show()
```

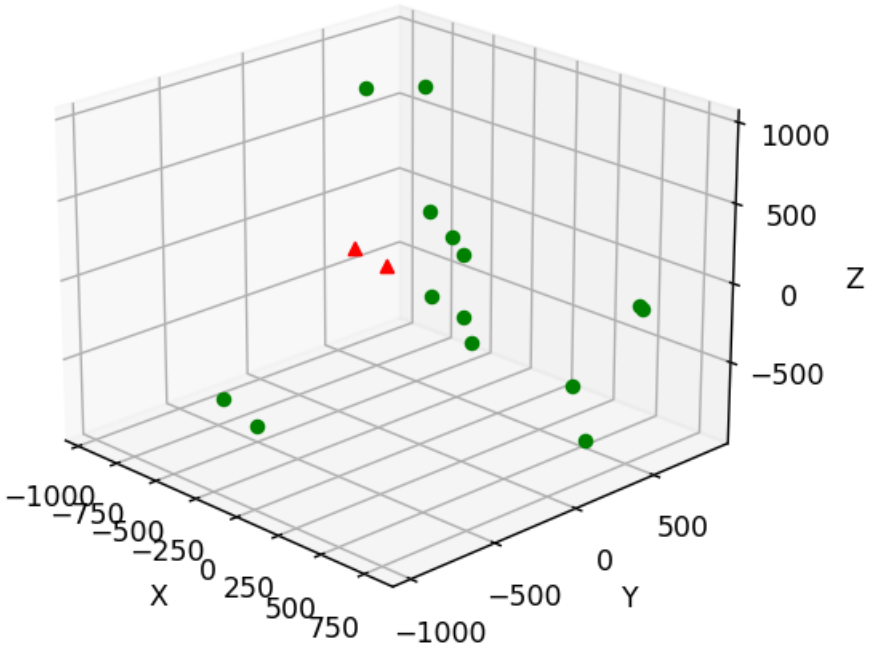
BAB IV EKSPERIMEN

4.1 Titik n = 16

```
PS C:\Users\alilo\OneDrive\Documents\KULIAH\Tingkat 2\Stima\Tucil2_13521128_13521146> & C:/Python310/python.exe "c:/Users/alilo/OneDrive/Do
21128_13521146/src/main.py"
Enter number of points generated: 16
Enter dimension of points generated: 3

Closest pair of points using Divide and Conquer:
((570.8162606742435, -970.3907578176554, 792.8160060419696), (630.9275164784679, -852.2639444688747, 668.6743478414778))
Distance :  181.59972028355742
13 times Euclidean distance calculated
Time taken:  0.000981569290161 seconds

Closest pair of points using Brute Force:
((570.8162606742435, -970.3907578176554, 792.8160060419696), (630.9275164784679, -852.2639444688747, 668.6743478414778))
Distance :  181.59972028355742
120 times Euclidean distance calculated
Time taken:  0.000000000000000 seconds
Device used: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```

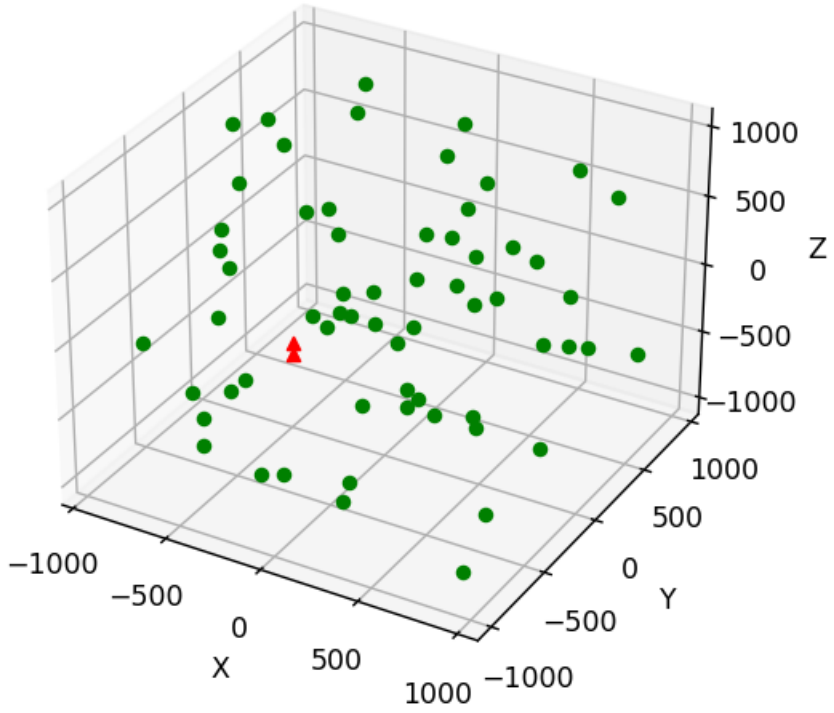


4.2 Titik n = 64

```
PS C:\Users\alilo\OneDrive\Documents\KULIAH\Tingkat 2\Stima\Tucil2_13521128_13521146> & C:/Python310/python.exe "c:/Users/alilo/OneDrive
21128_13521146/src/main.py"
Enter number of points generated: 64
Enter dimension of points generated: 3

Closest pair of points using Divide and Conquer:
((25.828331638524105, -862.6526709414077, 246.0682083268905), (28.280429302175207, -866.6253508076122, 326.2049596391587))
Distance :  80.27262222673274
64 times Euclidean distance calculated
Time taken:  0.001990556716919 seconds

Closest pair of points using Brute Force:
((25.828331638524105, -862.6526709414077, 246.0682083268905), (28.280429302175207, -866.6253508076122, 326.2049596391587))
Distance :  80.27262222673274
2016 times Euclidean distance calculated
Time taken:  0.004030942916870 seconds
Device used: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```

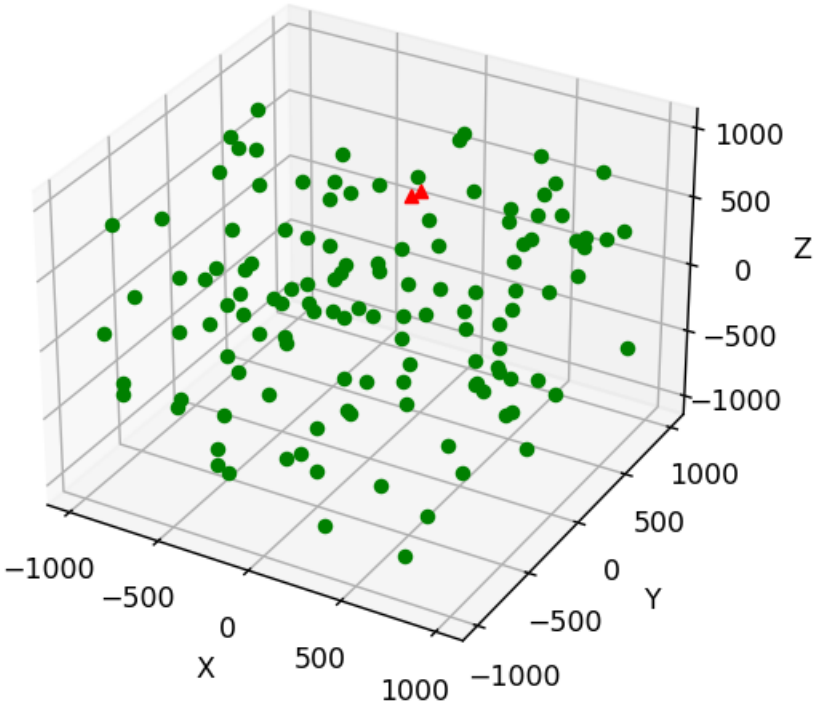


4.3 Titik n = 128

```
PS C:\Users\alilo\OneDrive\Documents\KULIAH\Tingkat 2\Stima\Tucil2_13521128_13521146> & C:/Python310/python.exe "c:/Users/alilo/OneDrive/D
21128_13521146/src/main.py"
Enter number of points generated: 128
Enter dimension of points generated: 3

Closest pair of points using Divide and Conquer:
((12.671798963993297, 381.97982277262645, 575.8149221583506), (50.12780036186791, 399.55292247549664, 614.8592037483495))
Distance :  56.887800087221976
124 times Euclidean distance calculated
Time taken:  0.005995512008667 seconds

Closest pair of points using Brute Force:
((12.671798963993297, 381.97982277262645, 575.8149221583506), (50.12780036186791, 399.55292247549664, 614.8592037483495))
Distance :  56.887800087221976
8128 times Euclidean distance calculated
Time taken:  0.017976760864258 seconds
Device used: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```

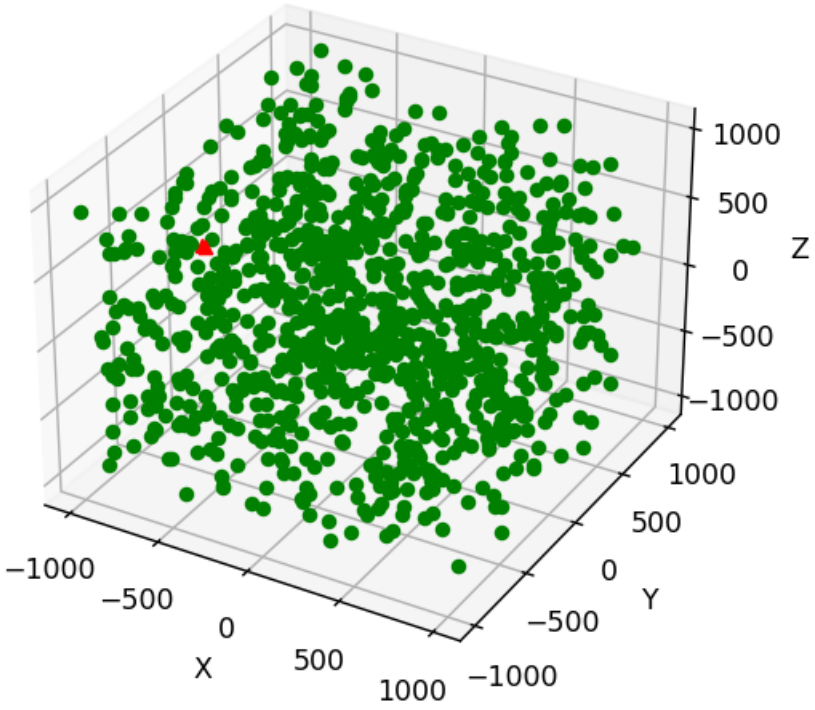


4.4 Titik n = 1000

```
PS C:\Users\alilo\OneDrive\Documents\KULIAH\Tingkat 2\Stima\Tucil2_13521128_13521146> & C:/Python310/python.exe "c:/Users/alilo/
21128_13521146/src/main.py"
Enter number of points generated: 1000
Enter dimension of points generated: 3

Closest pair of points using Divide and Conquer:
((-883.8309799106281, -86.41177675343647, 188.05386150513846), (-872.0337304819357, -87.91813701966271, 183.42767061655422))
Distance :  12.761107219676095
993 times Euclidean distance calculated
Time taken:  0.153548479080200 seconds

Closest pair of points using Brute Force:
((-883.8309799106281, -86.41177675343647, 188.05386150513846), (-872.0337304819357, -87.91813701966271, 183.42767061655422))
Distance :  12.761107219676095
499500 times Euclidean distance calculated
Time taken:  1.090216636657715 seconds
Device used: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```



4.5 Bonus 2 (5 Dimensi)

```
PS C:\Users\alilo\OneDrive\Documents\KULIAH\Tingkat 2\Stima\Tucil2_13521128_13521146> & C:/Python310/python.exe "c:/Users/alilo/OneDrive/Documents/KULIAH/Tingkat 2/Stima/Tucil2_135
21128_13521146/src/main.py"
Enter number of points generated: 200
Enter dimension of points generated: 5

Closest pair of points using Divide and Conquer:
((-421.5419165275475, 274.4291427960161, -307.0339997255438, -831.1447214576491, 260.06968388706287), (-378.7233533222818, 150.90348875990344, -306.8776760666308, -757.164241655456
1, 139.20953347186514))
Distance :  192.80126645654062
291 times Euclidean distance calculated
Time taken:  0.019993543624878 seconds

Closest pair of points using Brute Force:
((-421.5419165275475, 274.4291427960161, -307.0339997255438, -831.1447214576491, 260.06968388706287), (-378.7233533222818, 150.90348875990344, -306.8776760666308, -757.164241655456
1, 139.20953347186514))
Distance :  192.80126645654062
19900 times Euclidean distance calculated
Time taken:  0.058521509170532 seconds
Device used: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```

4.6 Bonus 2 (10 Dimensi)

```
PS C:\Users\alilo\OneDrive\Documents\KULIAH\Tingkat 2\Stima\Tucil2_13521128_13521146> & C:/Python310/python.exe "c:/Users/alilo/OneDrive/Documents/KULIAH/Tingkat 2/Stima/Tucil2_135
21128_13521146/src/main.py"
Enter number of points generated: 500
Enter dimension of points generated: 10

Closest pair of points using Divide and Conquer:
((-110.24412411822436, -367.6215141091608, -14.097161441003664, -734.324602711316, 107.2902880305046, -106.02237780033374, -702.664925822029, -459.6462657330527, 599.4424985123667,
  -888.3861856462123), (-77.52679615330703, -439.1372152131072, -74.98051577109857, -818.8199105947034, 61.279593466316555, -83.96379493561619, -772.1834350533596, -722.377342475476
3, 307.04647000051955, -947.2028455794097))
Distance :  427.1416659865929
2509 times Euclidean distance calculated
Time taken:  0.247208833694458 seconds

Closest pair of points using Brute Force:
((-110.24412411822436, -367.6215141091608, -14.097161441003664, -734.324602711316, 107.2902880305046, -106.02237780033374, -702.664925822029, -459.6462657330527, 599.4424985123667,
  -888.3861856462123), (-77.52679615330703, -439.1372152131072, -74.98051577109857, -818.8199105947034, 61.279593466316555, -83.96379493561619, -772.1834350533596, -722.377342475476
3, 307.04647000051955, -947.2028455794097))
Distance :  427.1416659865929
124750 times Euclidean distance calculated
Time taken:  0.599192380905151 seconds
Device used: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
```

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

BAB V KESIMPULAN DAN SARAN

5.1 Kesimpulan dan Saran

Pencarian jarak titik terdekat dengan strategi *divide and conquer* dan *brute force* dapat mencari hasil yang dijamin paling optimum. Dengan uji coba titik yang banyak hingga 1000 titik yang dibangkitkan dengan *randomisasi*, dapat dibuktikan bahwa program yang dibuat penulis dapat berjalan dengan cukup cepat dan melakukan operasi rumus jarak euclidian dengan efisien. Secara visual, program telah menunjukan *plot* 3D untuk masing-masing titik dan menandai sepasang titik dengan jarak terdekat dengan bantuan *library* matplotlib. Program juga telah berhasil untuk menggeneralisir persoalan pencarian jarak terdekat dalam dimensi ke-n untuk sekumpulan vektor di R^n. Penulis berharap adanya masukan dari pembaca sebab program dan laporan ini masih banyak kekurangan.

REFERENSI

Munir, Rinaldi. 2021. “Algoritma Divide and Conquer Bagian 2”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/stima22-23.htm, diakses pada 28 Februari 2023 pukul 17.34

LINK GITHUB

https://github.com/zakia215/Tucil2_13521128_13521146