Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

Penyelesaian Masalah Pencarian Titik Terdekat dengan Algoritma Divide and Conquer

Disusun Oleh:
Rava Maulana 13521149

Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2023

I. Deskripsi Persoalan

Masalah pencarian titik terdekat merupakan persoalan pencarian dua buah titik dengan jarak terdekat dari sekumpulan titik pada suatu ruang. Ruang yang didefinisikan untuk permasalah ini merupakan ruang vektor pada R^n. Penghitungan jarak dapat dilakukan menggunakan rumus euclidian pada tiga dimensi,

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

Rumus tersebut dapat digeneralisasi pada R^n, dengan mengurangkan setiap pasangan komponen vektor dari dua buah titik lalu menjumlahkan hasil kuadrat dari selisih setiap komponen vektor.

II. Algoritma

Algoritma yang digunakan dalam permasalahan ini merupakan jenis algoritma *divide and conquer* yang membagi-bagi suatu persoalan menjadi beberapa subpersoalan yang lebih kecil. Algoritma *divide and conquer* memiliki tiga langkah utama, yaitu *divide, solve,* dan *combine* dengan implementasinya sebagai berikut,

1. *Divide* :
   Membagi sekumpulan titik terurut menjadi dua buah daerah dengan jumlah titik yang setara.

2. *Solve* :
   Mencari dua buah titik terdekat pada setiap daerah secara rekursif.

3. *Conquer* :
   Menggabungkan hasil dari subpersoalan dengan membandingkan tiga kemungkinan: (asumsikan kumpulan titik awal dibagi menjadi daerah S_1 dan S_2)
   a. Kedua titik terdekat berada pada S_1.
   b. Kedua titik terdekat berada pada S_2.
   c. Salah satu titik berada pada S_1 dan titik lainnya berada pada S_2.

Berikut ini merupakan langkah-langkah dari algoritma pencarian titik terdekat secara *divide and conquer*,

1. Urutkan kumpulan titik awal berdasarkan koordinat x-nya secara menaik.
2. Kasus basis :
   Jika kumpulan titik hanya terdiri dari dua atau tiga buah titik, cari titik terdekat secara *brute force* dengan membandingkan semua titik.
   Kasus rekursif :

Bagi kumpulan titik menjadi dua daerah dengan jumlah titik yang setara, lalu cari titik terdekat dari setiap daerah.

3. Cari titik terdekat yang melintasi kedua buah daerah dengan menggunakan algoritma yang akan dijelaskan lebih lanjut di bawah.
4. Bandingkan jarak dari ketiga kemungkinan kasus, lalu cari jarak terkecilnya. Pasangan titik yang memiliki jarak terkcil tersebut merupakan pasangan titik terdekat dari sekumpulan titik awal.

Berikut ini merupakan algoritma yang digunakan untuk mencari titik terdekat yang melewati dua buah daerah S_1 dan S_2,

1. Tentukan titik terdekat pada S_1 dan S_2, lalu bandingkan kedua pasangan titik tersebut dan tentukan jarak terdekatnya (d).
2. Lakukan iterasi untuk semua titik di S_1 dan pasangkan satu-per-satu dengan semua titik pada S_2.
3. Periksa selisih dari setiap komponen yang ada pada kedua buah titik. Jika terdapat satu komponen dengan selisih yang lebih besar dari d, maka kedua buah titik bukanlah kandidat untuk pasangan titik terdekat, sehingga lanjutkan iterasi pada langkah 2.
4. Jika pemeriksaan pada langkah 3 berhasil, hitung jarak dari kedua buah titik.
5. Bandingkan jarak dari kedua buah titik dengan d, jika lebih kecil maka simpan sebagai kandidat pasangan titik terdekat.
6. Lanjutkan iterasi pada langkah 2.

III. Kode Program

Program dibuat menggunakan bahasa pemrograman *python*. Program teridiri dari satu *file* utama `main.py` dan sebuah folder `functions` yang terdiri dari *file* `brute_force.py`, `closest_pair.py`, `input_output.py`, `sorting.py`, `utility.py`, dan `visualizer.py`. Program utama terdiri dari sebuah fungsi `main` yang menggabungkan fungsi-fungsi dari folder `functions`.

```
ef main() → None:
 """Main function for this program"""
 num_points, dimension = get_user_input()
 points = get_random_points(num_points, dimension)

 sorted_points = quicksort_on_x(points)

 start_time_dnc = time()
 min_distance_dnc, point_1_dnc, point_2_dnc, count_euc_dnc = closest_pair(sorted_points)
 end_time_dnc = time()

 start_time_bf = time()
 min_distance_bf, point_1_bf, point_2_bf, count_euc_bf = brute_force(points)
 end_time_bf = time()

 exec_time_dnc = end_time_dnc - start_time_dnc
 exec_time_bf = end_time_bf - start_time_bf
 print_output(min_distance_dnc, point_1_dnc, point_2_dnc, count_euc_dnc, exec_time_dnc,
 min_distance_bf, point_1_bf, point_2_bf, count_euc_bf, exec_time_bf)

 if dimension == 3:
   visualize(points, point_1_dnc, point_2_dnc)
```

File `brute_force.py` berisi algoritma *brute force* yang akan digunakan untuk menguji kebenaran dari algoritma *divide and conquer*. *File* ini terdiri dari sebuah fungsi `brute_force` yang akan menerima masukan sekumpulan titik, lalu menghasilkan jarak terpendek beserta pasangan titik terdekat dari sekumpulan titik tersebut.

```
def brute_force(points: List[List[float]]) → Tuple[float, List[float], List[float], int]:
  count_euc = 0
  min_distance, point_1, point_2 = euclidean_distance(points[0], points[1]), points[0], points
  [1]

  for i in range(len(points)):
    for j in range(len(points)):
      temp_min = euclidean_distance(points[i], points[j])
      count_euc += 1

      if i ≠ j and temp_min < min_distance:
        min_distance, point_1, point_2 = temp_min, points[i], points[j]
```

File `closest_pair.py` berisi fungsi-fungsi yang digunakan dalam algoritma *divide and conquer*. Fungsi `is_in_strip` merupakan predikat yang menerima input dua buah titik beserta ukuran dari *strip*, lalu menghasilkan *true* jika dua titik tersebut berada pada *strip*. Fungsi `closest_pair_strip` menerima masukan sekumpulan titik, lalu menghasilkan jarak terpendek beserta pasangan titik terdekat yang melintasi dua daerah (melintasi *strip*). Fungsi

`closest_pair` menerima masukan sekumpulan titik, lalu menghasilkan jarak terpendek beserta pasangan titik terdekat dari sekumpulan titik tersebut.

```python
def is_in_strip(point_1: List[float], point_2: List[float], width: float) -> bool:
    for i in range(len(point_1)):
        if abs(point_1[i] - point_2[i] > width):
            return False
    return True
```

```python
def closest_pair_strip(left: List[List[float]], right: List[List[float]], width: float,
_point_1: List[float], _point_2: List[float]) -> Tuple[float, List[float], List[float]]:
    count_euc = 0
    min_distance, point_1, point_2 = width, _point_1, _point_2

    for point_left in left:
        for point_right in right:
            if is_in_strip(point_left, point_right, width):
                temp_distance = euclidean_distance(point_left, point_right)
                count_euc += 1

                if temp_distance < min_distance:
                    min_distance, point_1, point_2 = temp_distance, point_left, point_right

    return min_distance, point_1, point_2, count_euc
```

```python
def closest_pair(points: List[List[float]]) → Tuple[float, List[float], List[float], int]:
  if len(points) == 2:
    return euclidean_distance(points[0], points[1]), points[0], points[1], 1
  elif len(points) == 3:
    min_distance, index_1, index_2 = min3(euclidean_distance(points[0], points[1]),
                    euclidean_distance(points[0], points[2]),
                    euclidean_distance(points[1], points[2]))
    return min_distance, points[index_1], points[index_2], 3
  else:
    mid = len(points)//2
    left = points[:mid]
    right = points[mid:]

    min_left, point_left_1, point_left_2, count_left = closest_pair(left)
    min_right, point_right_1, point_right_2, count_right = closest_pair(right)

    min_distance, point_1, point_2 = (min_left, point_left_1, point_left_2) if min_left ⩽
    min_right else (min_right, point_right_1, point_right_2)

    min_strip, point_strip_1, point_strip_2, count_strip = closest_pair_strip(left, right,
    min_distance, point_1, point_2)

    min_distance, point_1, point_2 = (min_strip, point_strip_1, point_strip_2) if min_strip
    ⩽ min_distance else (min_distance, point_1, point_2)

    count_euc = count_left + count_right + count_strip
    return min_distance, point_1, point_2, count_euc
```

File `input_output.py` berisi fungsi-fungsi yang digunakan dalam menerima masukkan dan menampilkan hasil ke layar. Fungsi `get_user_input` menerima masukkan jumlah titik dan dimensi setiap titik dari pengguna. Fungsi `print_output` menampilkan data-data yang dibutuhkan ke layar. Fungsi `print_titik` digunakan untuk menampilkan sebuah titik ke layar dengan format tertentu.

```python
def get_user_input() → Tuple[int, int]:
  print(40*"=")
  print("TUGAS KECIL STRATEGI ALGORITMA")
  print(40*"=")

  num_points = int(input("Masukkan jumlah titik: "))
  dimension = int(input("Masukkan dimensi dari setiap titik: "))

  return num_points, dimension
```

```python
def print_output(min_distance_dnc: float, point_1_dnc: List[float], point_2_dnc: List[float],
count_ops_dnc: int, exec_time_dnc: float, min_distance_bf: float, point_1_bf: List[float],
point_2_bf: List[float], count_ops_bf: int, exec_time_bf: float) -> None:
  print("\n", 50*"=", sep="")
  print("ALGORITMA BRUTE FORCE")
  print("Jarak terpendek: ", min_distance_bf)
  print("Titik 1: ", end="")
  print_titik(point_1_bf)
  print("Titik 2: ", end="")
  print_titik(point_2_bf)
  print("Jumlah operasi euclidean distance: ", count_ops_bf)
  print(f"Waktu eksekusi algoritma: {exec_time_bf:.5f} detik")
  print(50*"=", "\n")

  print(50*"=")
  print("ALGORITMA DIVIDE AND CONQUER")
  print("Jarak terpendek: ", min_distance_dnc)
  print("Titik 1: ", end="")
  print_titik(point_1_dnc)
  print("Titik 2: ", end="")
  print_titik(point_2_dnc)
  print("Jumlah operasi euclidean distance: ", count_ops_dnc)
  print(f"Waktu eksekusi algoritma: {exec_time_dnc:.5f} detik")
  print(50*"=", "\n")
```

```python
def print_titik(point: List[float]) -> None:
  print("(", end="")
  for i in range(len(point)-1):
    print(f"{point[i]:.2f},", end="")
  print(f"{point[-1]:.2f})")
```

File `sorting.py` berisi fungsi-fungsi yang digunakan untuk melakukan pengurutan. Fungsi `partition` menerima sekumpulan titik, lalu menghasilkan partisi dari sekumpulan titik tersebut berdasarkan sebuah *pivot*. Fungsi `quicksort_on_x` menerima sekumpulan titik, lalu menghasilkan sekumpulan titik yang terurut menaik berdasarkan koordinat x-nya menggunakan algoritma pengurutan *quicksort*.

```python
def partition(_points: List[List[float]]) -> Tuple[List[List[float]], List[List[float]]]:
  points = _points[:]
  pivot = points[0]
  i = 1
  j = len(points)

  while i < j:
    if points[i][0] < pivot[0]:
      i += 1
    else:
      j -= 1
      points[i], points[j] = points[j], points[i]

  points[i-1], points[0] = points[0], points[i-1]
  return points[:i], points[i:]
```

```python
def quicksort_on_x(points: List[List[float]]) -> List[List[float]]:
  if len(points) <= 1:
    return points
  else:
    left, right = partition(points)
    return quicksort_on_x(left) + quicksort_on_x(right)
```

File `utility.py` berisi fungsi-fungsi pembantu yang dapat digunakan oleh fungsi lain. Fungsi `euclidean_distance` menerima masukan dua buah titik, lalu menghasilkan jarak euclidian dari kedua titik tersebut. Fungsi `min3` menerima masukan tiga bilangan bulat, lalu menghasilkan bilangan terkecil dari ketiga bilangan tersebut. Fungsi `get_random_points` menerima masukan jumlah titik dan dimensi setiap titik, lalu menghasilkan secara acak sebuah kumpulan titik.

```python
def euclidean_distance(point_1: List[float], point_2: List[float]) -> float:
  sum = 0
  for i in range(len(point_1)):
    sum += (point_1[i]-point_2[i])**2

  return sum**(1/2)
```

```python
def min3(f1: float, f2: float, f3: float) -> Tuple[float, int, int]:
  if f1 <= f2 and f1 <= f3 :
    return f1, 0, 1
  elif f2 <= f1 and f2 <= f3 :
    return f2, 0, 2
  else:
    return f3, 1, 2
```

```python
def get_random_points(num_points: int, dimension: int) -> List[List[float]]:
  return [[uniform(-500, 500) for _ in range(dimension)] for _ in range(num_points)]
```

File `visualizer.py` berisi fungsi untuk menampikan visualisasi dari hasil perhitungan algoritma. Fungsi `visualize` menerima masukan sekumpulan titik dan dua titik terdekat, lalu menampilkan semua titik ke layar.

```
def visualize(_points, point_1, point_2):
  points = _points[:]
  fig = plt.figure()

  axis = fig.add_subplot(111, projection='3d')
  for i in range(len(points)):
    axis.scatter(points[i][0], points[i][1], points[i][2], c='b', marker='o')

  axis.scatter(point_1[0], point_1[1], point_1[2], c='r', marker='o')
  axis.scatter(point_2[0], point_2[1], point_2[2], c='r', marker='o')

  axis.set_xlim([-500, 500])
  axis.set_ylim([-500, 500])
  axis.set_zlim([-500, 500])

  axis.set_xlabel('X')
  axis.set_ylabel('Y')
  axis.set_zlabel('z')

  plt.show()
```

IV. Percobaan

Berikut ini merupakan percobaan yang dilakukan untuk titik tiga dimensi dengan jumlah titik sebanyak 16, 64, 128, dan 1000, serta untuk titik sepuluh dimensi dengan jumlah titik sebanyak 16, 64, 128, dan 1000.

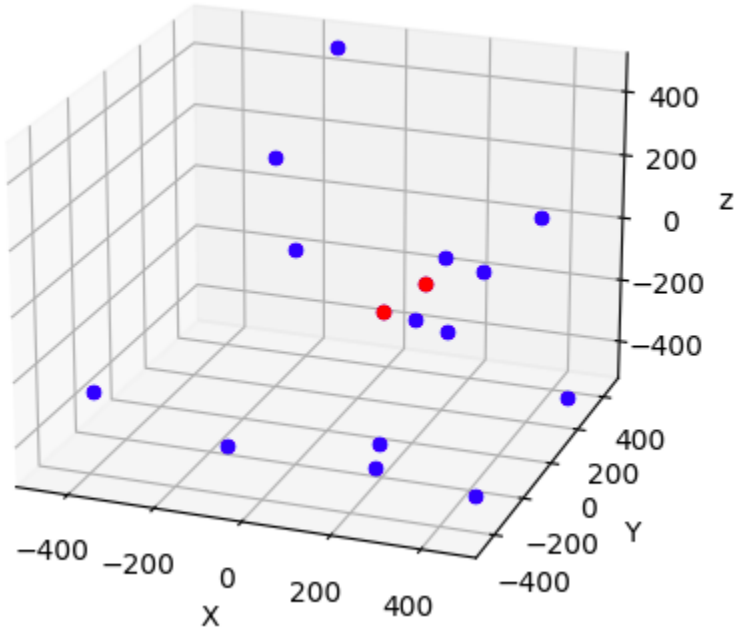
1. 16 Titik 3 Dimensi

```
rava@Ravas-MacBook-Air Tucil2_13521149 % python3 src/main.py
═══════════════════════════════════════
TUGAS KECIL STRATEGI ALGORITMA
═══════════════════════════════════════
Masukkan jumlah titik: 16
Masukkan dimensi dari setiap titik: 3
```

Input program untuk jumlah titik sebanyak 16

```
═══════════════════════════════════════════════════
ALGORITMA BRUTE FORCE
Jarak terpendek:  124.98946335090649
Titik 1: (154.74,215.73,-127.18)
Titik 2: (80.84,147.46,-201.34)
Jumlah operasi euclidean distance:  256
Waktu eksekusi algoritma: 0.00075 detik
═══════════════════════════════════════════════════


═══════════════════════════════════════════════════
ALGORITMA DIVIDE AND CONQUER
Jarak terpendek:  124.98946335090649
Titik 1: (80.84,147.46,-201.34)
Titik 2: (154.74,215.73,-127.18)
Jumlah operasi euclidean distance:  49
Waktu eksekusi algoritma: 0.00058 detik
═══════════════════════════════════════════════════
```

Output program untuk jumlah titik sebanyak 16



Hasil visualisasi program untuk jumlah titik sebanyak 16

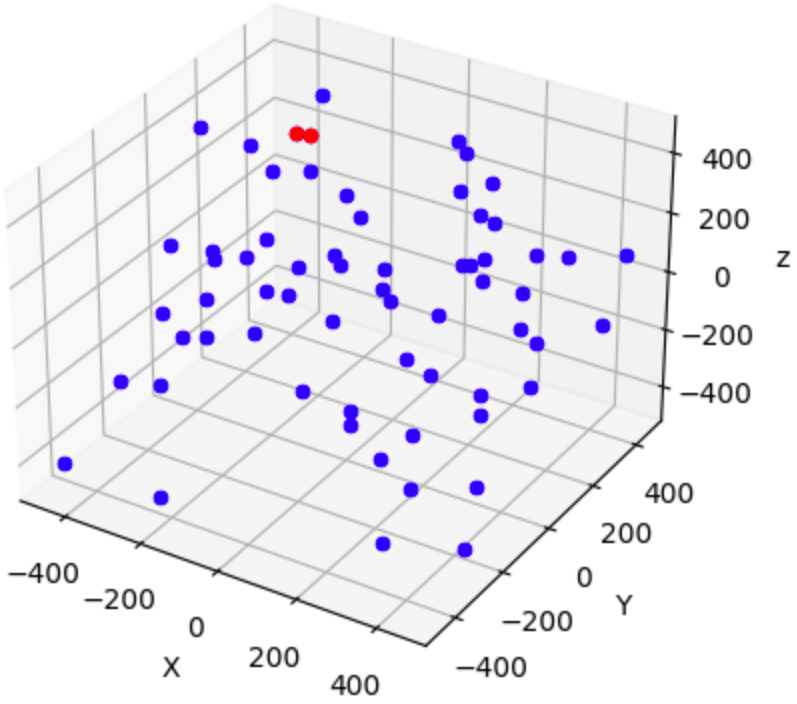
2. 64 Titik 3 Dimensi

```
rava@Ravas-MacBook-Air Tucil2_13521149 % python3 src/main.py
═══════════════════════════════════════════
TUGAS KECIL STRATEGI ALGORITMA
═══════════════════════════════════════════
Masukkan jumlah titik: 64
Masukkan dimensi dari setiap titik: 3
```

Input program untuk jumlah titik sebanyak 64

```
═══════════════════════════════════════════
ALGORITMA BRUTE FORCE
Jarak terpendek:  34.73805188534742
Titik 1: (-182.73,143.16,422.78)
Titik 2: (-217.27,139.57,421.73)
Jumlah operasi euclidean distance:  4096
Waktu eksekusi algoritma: 0.01003 detik
═══════════════════════════════════════════


═══════════════════════════════════════════
ALGORITMA DIVIDE AND CONQUER
Jarak terpendek:  34.73805188534742
Titik 1: (-217.27,139.57,421.73)
Titik 2: (-182.73,143.16,422.78)
Jumlah operasi euclidean distance:  687
Waktu eksekusi algoritma: 0.00568 detik
═══════════════════════════════════════════
```

Output program untuk jumlah titik sebanyak 64



Hasil visualisasi program untuk jumlah titik sebanyak 64

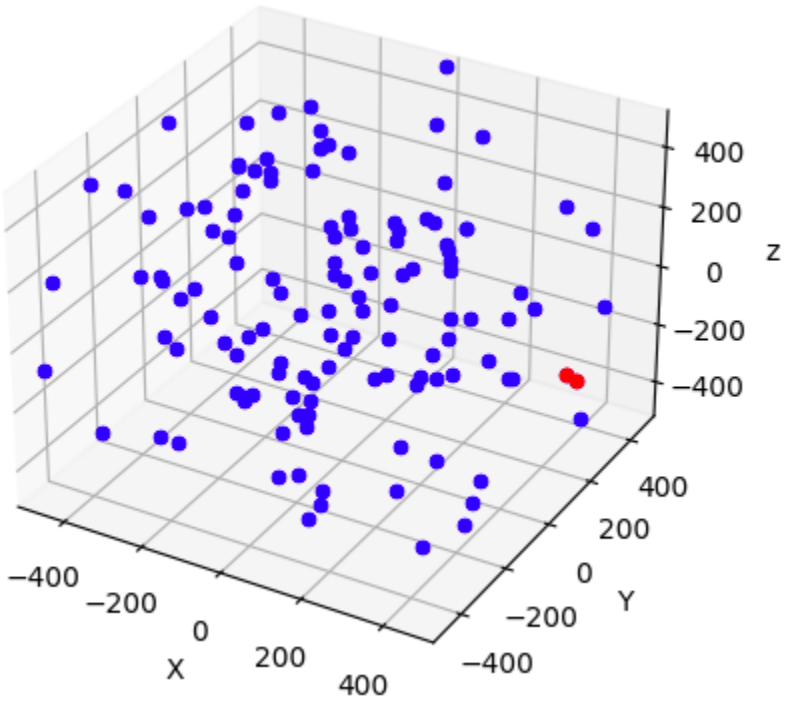
3. 128 Titik 3 Dimensi

```
rava@Ravas-MacBook-Air Tucil2_13521149 % python3 src/main.py
═══════════════════════════════════════════════
TUGAS KECIL STRATEGI ALGORITMA
═══════════════════════════════════════════════
Masukkan jumlah titik: 128
Masukkan dimensi dari setiap titik: 3
```

Input program untuk jumlah titik sebanyak 128

```
==========
ALGORITMA BRUTE FORCE
Jarak terpendek:  26.958977598962715
Titik 1: (456.42,238.96,-227.95)
Titik 2: (431.21,244.58,-220.21)
Jumlah operasi euclidean distance:  16384
Waktu eksekusi algoritma: 0.03518 detik
==========


==========
ALGORITMA DIVIDE AND CONQUER
Jarak terpendek:  26.958977598962715
Titik 1: (431.21,244.58,-220.21)
Titik 2: (456.42,238.96,-227.95)
Jumlah operasi euclidean distance:  2002
Waktu eksekusi algoritma: 0.01528 detik
==========
```

Output program untuk jumlah titik sebanyak 128



Hasil visualisasi program untuk jumlah titik sebanyak 128

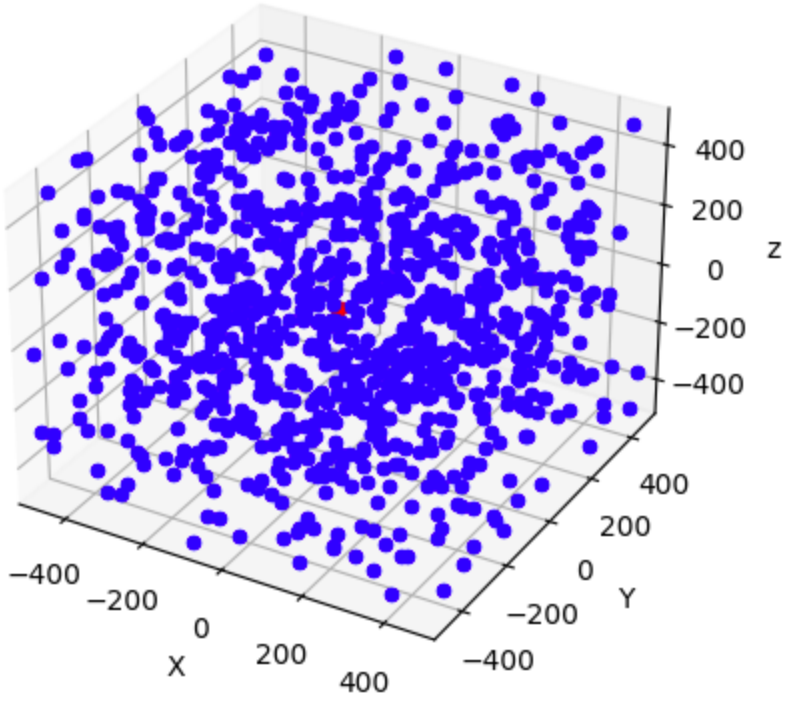
4. 1000 Titik 3 Dimensi

```
rava@Ravas-MacBook-Air Tucil2_13521149 % python3 src/main.py
═══════════════════════════════════════════════
TUGAS KECIL STRATEGI ALGORITMA
═══════════════════════════════════════════════
Masukkan jumlah titik: 1000
Masukkan dimensi dari setiap titik: 3
```

Input program untuk jumlah titik sebanyak 1000

```
═══════════════════════════════════════════════
ALGORITMA BRUTE FORCE
Jarak terpendek:  8.6429925319234
Titik 1: (28.50,-76.05,59.66)
Titik 2: (32.98,-70.13,55.24)
Jumlah operasi euclidean distance:  1000000
Waktu eksekusi algoritma: 2.09229 detik
═══════════════════════════════════════════════


═══════════════════════════════════════════════
ALGORITMA DIVIDE AND CONQUER
Jarak terpendek:  8.6429925319234
Titik 1: (28.50,-76.05,59.66)
Titik 2: (32.98,-70.13,55.24)
Jumlah operasi euclidean distance:  130101
Waktu eksekusi algoritma: 0.99333 detik
═══════════════════════════════════════════════
```

Output program untuk jumlah titik sebanyak 1000



Hasil visualisasi program untuk jumlah titik sebanyak 1000

5. 16 Titik 10 Dimensi

```
[rava@Ravas-MacBook-Air Tucil2_13521149 % python3 src/main.py
═══════════════════════════════════════════════
TUGAS KECIL STRATEGI ALGORITMA
═══════════════════════════════════════════════
Masukkan jumlah titik: 16
Masukkan dimensi dari setiap titik: 16
```

Input program untuk jumlah titik sebanyak 16

```
═══════════════════════════════════════════════════
ALGORITMA BRUTE FORCE
Jarak terpendek:  863.9276844032821
Titik 1: (-0.83,225.38,413.09,-167.19,353.37,-233.88,-289.51,2.01,271.98,193.75,
304.05,-252.46,-383.14,385.05,-330.60,166.55)
Titik 2: (-31.84,-97.80,180.24,-94.51,21.13,-265.47,63.20,366.41,-77.03,108.43,1
46.73,-392.31,-315.08,485.63,-183.28,111.44)
Jumlah operasi euclidean distance:  256
Waktu eksekusi algoritma: 0.00156 detik
═══════════════════════════════════════════════════


═══════════════════════════════════════════════════
ALGORITMA DIVIDE AND CONQUER
Jarak terpendek:  863.9276844032821
Titik 1: (-31.84,-97.80,180.24,-94.51,21.13,-265.47,63.20,366.41,-77.03,108.43,1
46.73,-392.31,-315.08,485.63,-183.28,111.44)
Titik 2: (-0.83,225.38,413.09,-167.19,353.37,-233.88,-289.51,2.01,271.98,193.75,
304.05,-252.46,-383.14,385.05,-330.60,166.55)
Jumlah operasi euclidean distance:  113
Waktu eksekusi algoritma: 0.00158 detik
═══════════════════════════════════════════════════
```

Output program untuk jumlah titik sebanyak 16

6. 64 Titik 10 Dimensi

```
rava@Ravas-MacBook-Air Tucil2_13521149 % python3 src/main.py
═══════════════════════════════════════
TUGAS KECIL STRATEGI ALGORITMA
═══════════════════════════════════════
Masukkan jumlah titik: 64
Masukkan dimensi dari setiap titik: 10
```

Input program untuk jumlah titik sebanyak 64

```
==========
ALGORITMA BRUTE FORCE
Jarak terpendek:  494.06099870122483
Titik 1: (17.41,230.56,-334.07,-93.88,-98.53,-274.23,197.54,-56.93,-62.32,464.93
)
Titik 2: (125.38,230.93,-319.67,-191.49,156.11,-169.97,2.19,-234.17,75.52,223.23
)
Jumlah operasi euclidean distance:  4096
Waktu eksekusi algoritma: 0.01825 detik
==========


==========
ALGORITMA DIVIDE AND CONQUER
Jarak terpendek:  494.06099870122483
Titik 1: (17.41,230.56,-334.07,-93.88,-98.53,-274.23,197.54,-56.93,-62.32,464.93
)
Titik 2: (125.38,230.93,-319.67,-191.49,156.11,-169.97,2.19,-234.17,75.52,223.23
)
Jumlah operasi euclidean distance:  843
Waktu eksekusi algoritma: 0.00973 detik
==========
```

Output program untuk jumlah titik sebanyak 64

7. 128 Titik 10 Dimensi

```
[rava@Ravas-MacBook-Air Tucil2_13521149 % python3 src/main.py
==========
TUGAS KECIL STRATEGI ALGORITMA
==========
Masukkan jumlah titik: 128
Masukkan dimensi dari setiap titik: 10
```

Input program untuk jumlah titik sebanyak 128

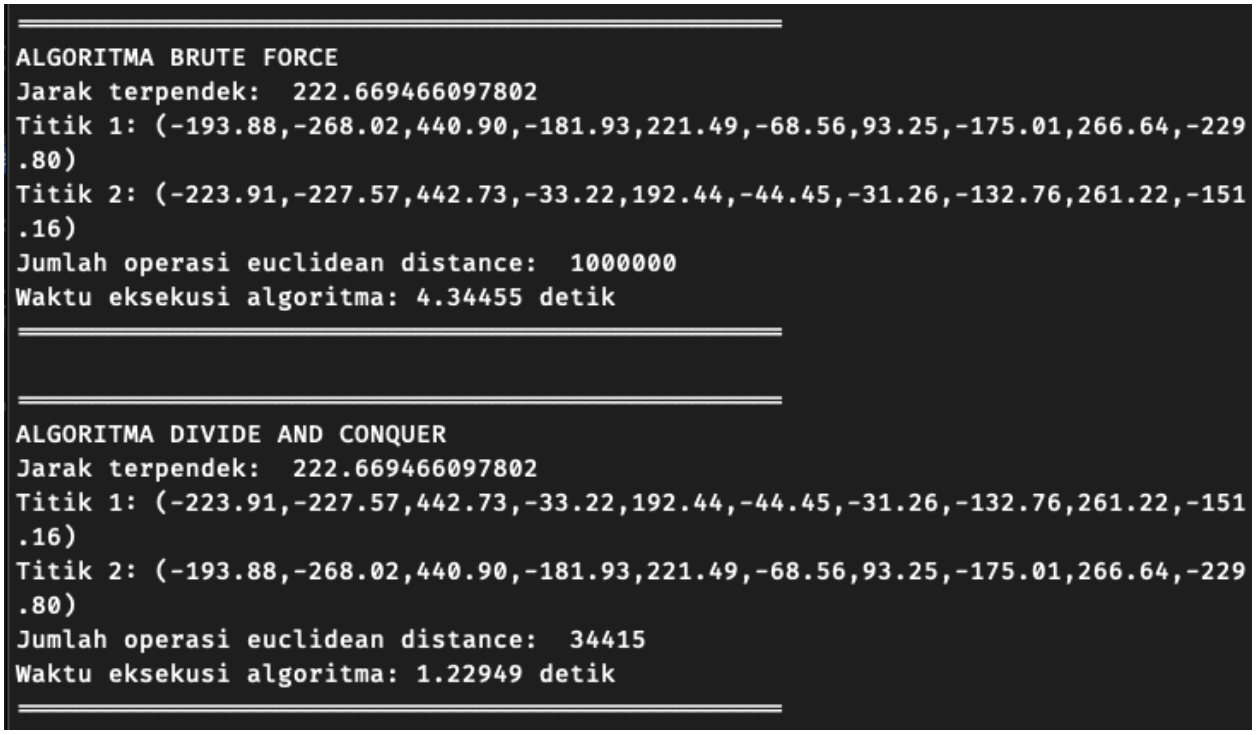
```
═══════════════════════════════════════════════════
ALGORITMA BRUTE FORCE
Jarak terpendek:  399.0237771213525
Titik 1: (134.89,487.77,-248.10,-85.70,-272.92,162.07,375.99,-120.47,-46.85,-487
.12)
Titik 2: (129.83,315.76,-265.19,-231.61,-191.71,267.42,183.07,-217.71,136.42,-38
6.75)
Jumlah operasi euclidean distance:  16384
Waktu eksekusi algoritma: 0.07189 detik
═══════════════════════════════════════════════════


═══════════════════════════════════════════════════
ALGORITMA DIVIDE AND CONQUER
Jarak terpendek:  399.0237771213525
Titik 1: (129.83,315.76,-265.19,-231.61,-191.71,267.42,183.07,-217.71,136.42,-38
6.75)
Titik 2: (134.89,487.77,-248.10,-85.70,-272.92,162.07,375.99,-120.47,-46.85,-487
.12)
Jumlah operasi euclidean distance:  1838
Waktu eksekusi algoritma: 0.02532 detik
═══════════════════════════════════════════════════
```

Output program untuk jumlah titik sebanyak 128

8. 1000 Titik 10 Dimensi

```
[rava@Ravas-MacBook-Air Tucil2_13521149 % python3 src/main.py
 ═══════════════════════════════════════
TUGAS KECIL STRATEGI ALGORITMA
 ═══════════════════════════════════════
Masukkan jumlah titik: 1000
Masukkan dimensi dari setiap titik: 10
```

Input program untuk jumlah titik sebanyak 1000



Output program untuk jumlah titik sebanyak 1000

V. Lampiran

Link Github: https://github.com/RMA1403/Tucil2_13521149

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