

**LAPORAN TUGAS KECIL 2 IF2211 STRATEGI ALGORITMA**

**MENCARI PASANGAN TITIK TERDEKAT PADA BIDANG TIGA  
DIMENSI DENGAN ALGORITMA DIVIDE AND CONQUER**

**Dosen : Dr. Ir. Rinaldi, M.T.**



**Disusun oleh :**

**Muhammad Hanan      13521041**

**Saddam Annais S      13521121**

**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**2022/2023**

## **Daftar Isi**

<b>Daftar Isi</b>	<b>1</b>
<b>BAB I : DESKRIPSI MASALAH</b>	<b>2</b>
<b>BAB II : ALGORITMA</b>	<b>3</b>
<b>BAB III : SOURCE CODE</b>	<b>7</b>
<b>BAB IV : TEST CASE</b>	<b>10</b>
<b>BAB V : TABLE</b>	<b>14</b>
<b>Lampiran</b>	<b>15</b>

## **BAB**

### **DESKRIPSI MASALAH**

Mencari sepasang titik terdekat dengan Algoritma Divide and Conquer sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Pada Tugil 2 kali ini Anda diminta 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  $P = (x, y, z)$ . Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titik  $P_1 = (x_1, y_1, z_1)$  dan  $P_2 = (x_2, y_2, z_2)$  dihitung dengan rumus Euclidean berikut:

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

Buatlah program dalam Bahasa C/C++/Java/Python/Golang/Ruby/Perl (pilih salah satu) untuk mencari sepasang titik yang jaraknya terdekat satu sama lain dengan menerapkan algoritma divide and conquer untuk penyelesaiannya, dan perbandingannya dengan Algoritma Brute Force.

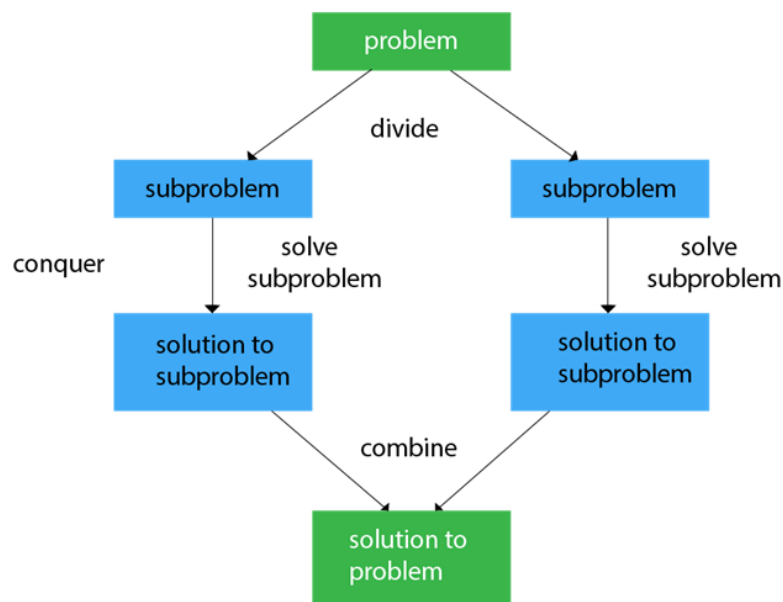
## BAB II

### ALGORITMA

Algoritma yang akan digunakan dalam mencari jarak minimum dari titik bidang 3 dimensi yaitu algoritma Divide and Conquer.

#### A. Algoritma Divide and Conquer

Algoritma divide and conquer adalah teknik pemecahan masalah yang menguraikan masalah menjadi beberapa submasalah yang lebih kecil, kemudian menyelesaikan setiap submasalah secara terpisah, dan terakhir menggabungkan solusi dari setiap submasalah untuk mendapatkan solusi akhir dari masalah asal. Algoritma ini terdiri dari tiga tahap, yaitu

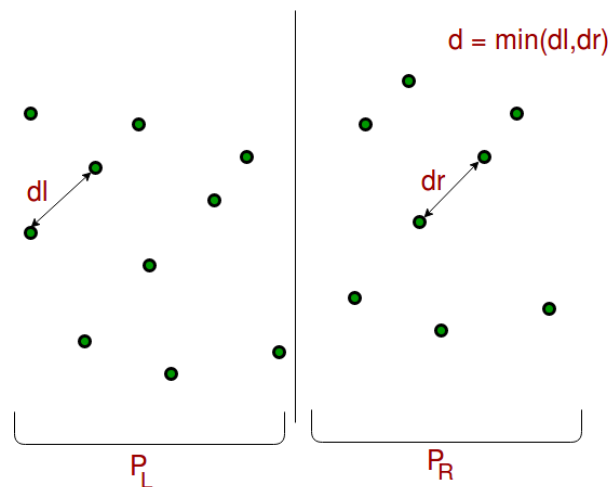


1. Divide, membagi masalah menjadi submasalah yang lebih kecil. Biasanya menjadi setengahnya atau mendekati setengahnya.
2. Conquer, ketika sebuah submasalah sudah cukup kecil untuk diselesaikan, langsung selesaikan masalah tersebut.
3. Combine: Solusi dari setiap submasalah digabungkan untuk menghasilkan solusi akhir dari masalah asal.

## B. Penerapan Algoritma Divide and Conquer pada Program

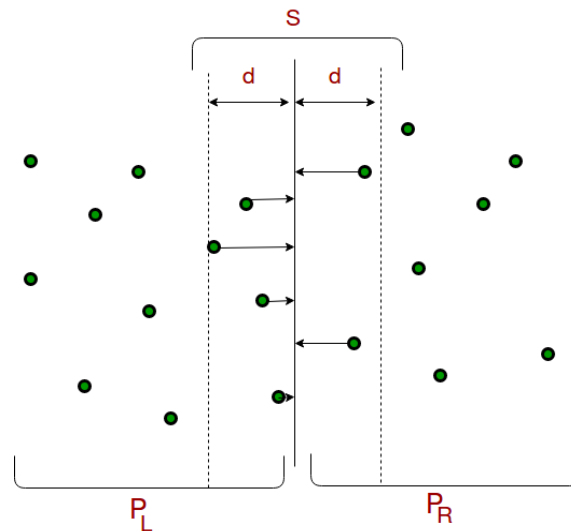
Algoritma untuk mencari pasangan titik terdekat pada tiga dimensi dan bahkan  $n$  dimensi dapat dilakukan dengan cara memodifikasi algoritma mencari pasangan titik terdekat pada dua dimensi. Adapun langkah-langkah mencari pasangan titik terdekat pada dua dimensi adalah sebagai berikut:

1. Setelah himpunan titik-titik pada bidang dibuat, semua titik akan diurutkan berdasarkan sumbu X. Urutan sorting dibebaskan dapat secara naik maupun turun, hal yang penting adalah titik-titik tersebut berurutan.
2. Kemudian, himpunan titik-titik tersebut dibagi menjadi dua bagian tepat di tengah menjadi subhimpunan bagian kiri dan subhimpunan bagian kanan.
3. Subhimpunan titik-titik tersebut akan terus dibagi menjadi dua secara rekursif sampai titik-titik di subhimpunan itu bersisa dua atau tiga elemen.
4. Jika subhimpunan bersisa dua atau tiga elemen, himpunan titik tersebut akan dicari pasangan titik terdekatnya menggunakan brute force. Hasil pasangan terdekat ini hanya untuk subhimpunan tersebut.
5. Untuk menggabungkan pasangan titik terdekat pada dua subhimpunan, subhimpunan kiri dan subhimpunan kanan dibandingkan dan hasil dari perbandingan tersebut merupakan pasangan titik dengan jarak yang lebih kecil.



6. Kemudian, hasil dari penyelesaian ini dibandingkan kembali dengan subhimpunan baru. Subhimpunan baru ini berisi semua titik yang jarak ke garis pemisah vertikal antara dua subhimpunan kurang dari jarak pasangan titik terdekat saat ini. Garis pemisah ini biasa

disebut dengan "strip". Hasil dari perbandingan ini adalah titik dengan jarak yang paling kecil.



7. Setelah itu, algoritma akan terus menerus combine sampai ditemukan pasangan titik yang paling dekat.

Untuk memodifikasi algoritma sehingga dapat mencari pasangan titik terdekat dapat dilakukan dengan merubah beberapa step. Pada algoritma dua dimensi, pembagian subhimpunan hanya berdasarkan sumbu X saja. Pada tiga dimensi, pembagian subhimpunan dapat dilakukan berdasarkan sumbu X dan Y. Pola ini juga dapat diterapkan pada dimensi yang lebih tinggi. Jika  $n$  merupakan banyaknya dimensi, subhimpunannya perlu dibagi berdasarkan sumbu  $e_1, e_2, \dots$  dan  $e_{(n-1)}$ . Pembagian subhimpunan berdasarkan sumbu dilakukan secara bergantian sampai jumlah titik di dalam subhimpunan adalah dua atau tiga. Pembagian subhimpunan berdasarkan mengikuti rumus  $s = s \% n$  dengan  $s$  adalah sumbu yang menjadi acuan pembagian.

Contohnya, pada divide pertama akan diurutkan berdasarkan sumbu  $e_1$ . Jika jumlah titik pada subhimpunan tersebut lebih dari tiga, subhimpunan akan dibagi lagi berdasarkan sumbu  $e_2$ . Pembagian subhimpunan akan dilakukan terus menerus sampai jumlah titik di dalam subhimpunan adalah dua atau tiga.

Tahap combine pada subhimpunan titik-titik tiga dimensi dan dimensi yang lebih tinggi memiliki konsep yang sama seperti combine pada dua dimensi. Konsepnya yaitu membuat subhimpunan dari titik-titik yang jarak ke strip lebih kecil dari jarak pasangan terpendek saat ini. Kemudian membandingkan jarak-jarak antar titik pada himpunan tersebut

dengan jarak pasangan terpendek saat ini. Setelah mendapatkan ide dari cara mencari pasangan titik terdekat pada bidang tiga atau lebih dimensi, kita dapat menuliskan langkah-langkahnya menjadi lebih rinci sebagai berikut:

1. Misal  $n$  merupakan banyaknya dimensi yang ingin dibuat dengan jumlah  $n \geq 1$ . Setelah himpunan titik-titik pada  $n$  dimensi dibuat, semua titik akan diurutkan berdasarkan sumbu  $e_1, e_2, \dots$  atau  $e_3$ . Pengurutan sumbu dilakukan secara bergantian dengan rumus  $s = (s + 1) \% n$  setiap pemanggilan fungsi.
2. Kemudian, himpunan titik-titik tersebut dibagi menjadi dua bagian tepat di tengah sumbu  $s$  menjadi subhimpunan bagian kiri dan subhimpunan bagian kanan.
3. Subhimpunan titik-titik tersebut akan terus dibagi menjadi dua dengan bergantian pada sumbu  $s$  sampai titik-titik di subhimpunan itu bersisa dua atau tiga elemen.
4. Jika subhimpunan bersisa dua atau tiga elemen, himpunan titik tersebut akan dicari pasangan titik terdekatnya menggunakan brute force. Hasil pasangan terdekat ini hanya untuk subhimpunan tersebut.
5. Untuk menggabungkan pasangan titik terdekat pada dua subhimpunan, subhimpunan kiri dan subhimpunan kanan dibandingkan dan hasil dari perbandingan tersebut merupakan pasangan titik dengan jarak yang lebih kecil.
6. Kemudian, hasil dari penyelesaian ini dibandingkan kembali dengan subhimpunan baru. Subhimpunan baru ini berisi semua titik yang jarak ke garis strip kurang dari jarak pasangan titik terdekat saat ini. Hasil dari perbandingan ini adalah titik dengan jarak yang paling kecil.
7. Setelah itu, algoritma akan terus menerus combine sampai ditemukan pasangan titik yang paling dekat.

## BAB III

### SOURCE CODE

#### A. main.py

```
1 import random
2 import os.path
3 from visualizer_3d import *
4 from closest_point_nd import welcome
5 from closest_point_nd import readPoints
6 from closest_point_nd import getClosestPoint
7 from closest_point_nd import printClosestPoint
8
9 welcome()
10 inp1 = int(input("\nInput Terminal      : 1\nInput file      : 2\n\033[33m>>> \033[0m"))
11
12 n = 0
13 p = 0
14 check = True
15 if inp1 == 1 :
16     n = int(input("\nNumber of Dimensions > 0      : "))
17     p = int(input("\nNumber of Points > 1      : "))
18     points = [tuple(random.uniform(-10000, 10000) for i in range(n)) for i in range(p)]
19 elif inp1 == 2 :
20     path = "../test/"
21     nameFile = input("\nInput the file name      : ")
22     if os.path.isfile(path+nameFile):
23         n, p, check, points = readPoints(path+nameFile)
24     else :
25         print("\033[31m+"\nFile doesnt exist. Exiting...\n"+" \033[0m")
26         exit()
27 else :
28     print("\033[31m+"\nInvalid input. Exiting...\n"+" \033[0m")
29     exit()
31 if (n < 1 or p < 2 or check == False):
32     print("\033[31m+"\nInvalid input. Exiting...\n"+" \033[0m")
33 else :
34     solution = getClosestPoint(points)
35     printClosestPoint("Divide and Conquer", solution[0], solution[2], solution[4], solution[6], solution[7])
36     printClosestPoint("Brute Force", solution[1], solution[3], solution[5], solution[7], solution[8])
37     print()
38
39 if (n == 3):
40     inp2 = int(input("Want to visualized?\nYes : 1\nNo : 0\n\033[33m>>> \033[0m"))
41     print()
42     if inp2 == 1 :
43         visualizePoints3D(points, solution[0], solution[4])
44     print("\033[31m+"\nExiting...\n"+" \033[0m")
```

File ini merupakan file yang akan di-run untuk menjalankan keseluruhan program. Nantinya program akan memberikan pilihan ingin input file atau input dengan di command line yang nantinya akan meminta inputan jumlah dimensi dan jumlah titik yang ingin dibangun pada program ini.

#### B. visualizer\_3d.py

```
import matplotlib.pyplot as plt

def visualizePoints3D(points, pair, distance):
    n = len(points)
    x = [points[i][0] for i in range(n)]
    y = [points[j][1] for j in range(n)]
    z = [points[k][2] for k in range(n)]

    fig = plt.figure(figsize=(6,6))
    fig.suptitle('Closest Point', fontsize=16, fontweight='bold')
    ax = plt.axes(projection="3d")
    fg = ax.scatter3D(x, y, z, alpha = 0.4)
    fg = ax.scatter3D(pair[0][0], pair[0][1], pair[0][2], alpha = 1, color='r')
    fg = ax.scatter3D(pair[1][0], pair[1][1], pair[1][2], alpha = 1, color='r')
    ax.set_title(f"Distance : {(distance):.5f}", loc = "left", fontsize = 9)
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    plt.show()
```

Pada file ini terdapat sebuah fungsi yang digunakan untuk memvisualisasikan titik yang dibangun pada bidang 3D.



### C. closest\_point\_nd.py

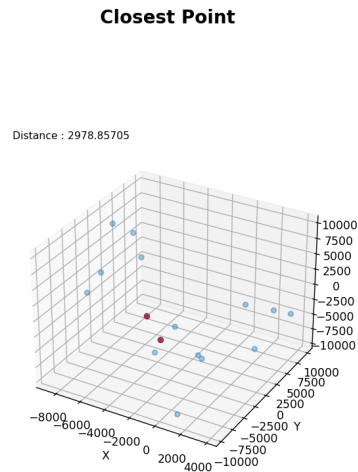
<pre>def euclDist(p1, p2, numCompute):     n = len(p1)     return math.sqrt(sum([(p1[i]-p2[i])**2 for i in range(n)])), numCompute + 1</pre>	<p>Fungsi ini digunakan untuk menghitung nilai euclidean dari 2 buah titik.</p>
<pre>def sortPair(points, idx):     points = sorted(points, key=lambda x: x[idx])     return points</pre>	<p>Fungsi ini digunakan untuk menyortir point berdasarkan dimensinya.</p>
<pre>def closestPairBF(points, numCompute):     n = len(points)     shortestDist = math.inf     closestPair = None     for i in range(n):         for j in range(i+1, n):             distance, numCompute = euclDist(points[i], points[j], numCompute)             if distance &lt; shortestDist:                 shortestDist = distance                 closestPair = (points[i], points[j])     return closestPair, shortestDist, numCompute</pre>	<p>Fungsi ini digunakan untuk mencari titik terdekat dengan menggunakan algoritma brute force</p>
<pre>def closestInStrip(shortestDist, closestPair, leftHalf, rightHalf, idx, numCompute):     mid = (leftHalf[-1][idx] + rightHalf[0][idx]) / 2      for p in leftHalf:         if (abs(mid - p[idx]) &lt; shortestDist):             for q in rightHalf:                 if (abs(mid - q[idx]) &lt; shortestDist):                     qualified = True                      for i in range(len(p)):                         if i != idx:                             if abs(q[i] - p[i]) &gt; shortestDist:                                 qualified = False                      if qualified:                         d, numCompute = euclDist(p, q, numCompute)                         if d &lt; shortestDist:                             shortestDist = d                             closestPair = (p, q)     return closestPair, shortestDist, numCompute</pre>	<p>Fungsi ini digunakan untuk mengecek apakah titik yang paling kanan dari himpunan daerah kiri dengan titik yang paling kiri dari himpunan daerah kanan itu jaraknya lebih pendek dari pada shortestDist. Jika iya maka nilai shortestDist akan diperbarui.</p>

<pre>def closestPairDnC(points, idx, numCompute):     # Basis     if (len(points) &lt;= 3):         return closestPairBF(points, numCompute)      # Rekurens     idx = (idx + 1) % len(points[0])     shortestDist = math.inf     closestPair = None      points = sortPair(points, idx)     mid = len(points)//2     leftHalf = points[:mid]     rightHalf = points[mid:]      pairAtLeft, distAtLeft, numCompute = closestPairDnC(leftHalf, idx, numCompute)     pairAtRight, distAtRight, numCompute = closestPairDnC(rightHalf, idx, numCompute)      if (distAtLeft &lt; distAtRight):         shortestDist = distAtLeft         closestPair = pairAtLeft     else:         shortestDist = distAtRight         closestPair = pairAtRight      closestPair, shortestDist, numCompute = closestInStrip(         shortestDist, closestPair, leftHalf, rightHalf, idx, numCompute)     return closestPair, shortestDist, numCompute</pre>	<p>Fungsi ini merupakan fungsi utama untuk mencari 2 buah titik terdekat secara Divide and Conquer</p>
<pre>def printClosestPoint(type, pair, numCompute, distance, time1, time2):     print("\n"+"033[36m"+ type + "\033[0m")     print(f"Point 1           : {pair[0]}")     print(f"Point 2           : {pair[1]}")     print(f"Distance            : {distance}")     print(f"Number of Euclidian Compute : {numCompute}")     executionTime = (time2 - time1)*10**6     if executionTime &gt; 500:         executionTime /= 1000         print(f"Time Execution           : {(executionTime):.3f} mili seconds")     else :         print(f"Time Execution           : {(executionTime):.3f} micro seconds")</pre>	<p>Fungsi ini adalah fungsi untuk mengeluarkan output dari hasil yang didapatkan.</p>
<pre>def readPoints(filePath):     with open(filePath, 'r') as file:         # Membaca baris pertama untuk mendapatkan besar dimensi dan jumlah titik         dimensions = tuple(map(int, file.readline().split()))         tuples = ()         check = True         for i in range(dimensions[1]):             rowValues = tuple(map(float, file.readline().replace(',', ' ').split()))             if len(rowValues) != dimensions[0] :                 check = False             tuples += rowValues,         return dimensions[0], dimensions[1], check, tuples</pre>	<p>Fungsi yang dipanggil ketika user memilih memasukkan inputan melalui file txt. Fungsi ini akan membaca file yang berisi points dan mengubahnya ke tuples</p>
<pre>def getClosestPoint(points):     time1 = time.perf_counter()     pairDnC, distDnC, numComputeDnC = closestPairDnC(points, -1, 0)     time2 = time.perf_counter()     pairBF, distBF, numComputeBF = closestPairBF(points, 0)     time3 = time.perf_counter()     return [pairDnC, pairBF, numComputeDnC, numComputeBF, distDnC, distBF, time1, time2, time3]</pre>	<p>Fungsi ini adalah fungsi yang mengambil hasil dari pemanggilan fungsi BF dan DnC untuk dilakukan output dan visualisasi</p>

## BAB IV

### TEST CASE

1. Saat bidang 3D
  - a. Jumlah titik 16



```
PS C:\Users\mmuuh\Documents\Tucil2-Stima\src> python main.py



Input Terminal      : 1
Input file          : 2
>>> 1

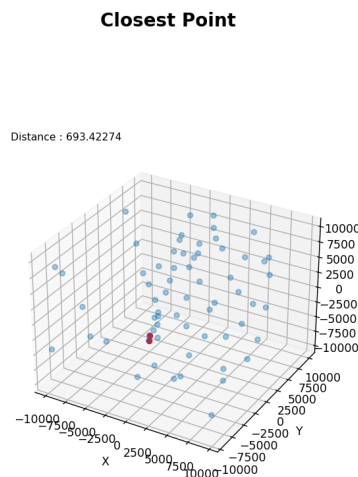
Number of Dimensions > 0 : 3
Number of Points > 1 : 16

Divide and Conquer
Point 1              : (-3775.1683772164244, -2750.661324879988, -1010.7390919571699)
Point 2              : (-2012.9068618162064, -4503.730977729728, -2652.3232773726413)
Distance             : 2978.8570460655246
Number of Euclidian Compute : 20
Time Execution       : 0.554 mili seconds

Brute Force
Point 1              : (-2012.9068618162064, -4503.730977729728, -2652.3232773726413)
Point 2              : (-3775.1683772164244, -2750.661324879988, -1010.7390919571699)
Distance             : 2978.8570460655246
Number of Euclidian Compute : 120
Time Execution       : 0.666 mili seconds

Want to visualized?
Yes : 1
No  : 0
>>> 1
```

- b. Jumlah titik 64



```


Input Terminal      : 1
Input file          : 2
>>> 1

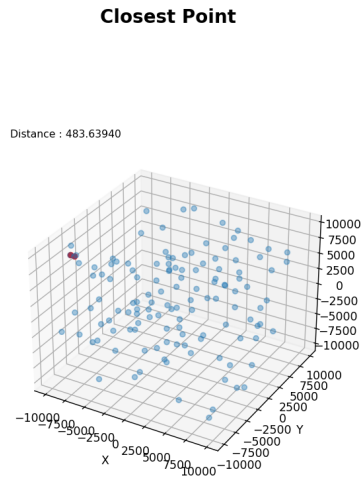
Number of Dimensions > 0 : 3
Number of Points > 1 : 64

Divide and Conquer
Point 1              : (-3107.7524545751767, -701.0036949597506, -6150.762967723697)
Point 2              : (-3199.446167441005, -389.5414543122588, -5538.04881547797)
Distance             : 693.4227402461488
Number of Euclidian Compute : 52
Time Execution       : 1.154 mili seconds

Brute Force
Point 1              : (-3107.7524545751767, -701.0036949597506, -6150.762967723697)
Point 2              : (-3199.446167441005, -389.5414543122588, -5538.04881547797)
Distance             : 693.4227402461488
Number of Euclidian Compute : 2016
Time Execution       : 11.463 mili seconds

Want to visualized?
Yes : 1
No  : 0
>>> 1
```

c. Jumlah titik 128



```


Closest Pair  
of Points


  Input Terminal      : 1
  Input file         : 2
  >>> 1

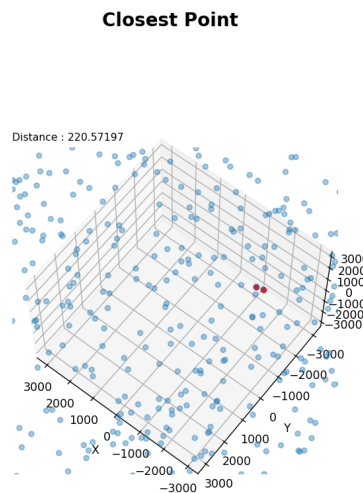
  Number of Dimensions > 0 : 3
  Number of Points > 1    : 128

  Divide and Conquer
  Point 1                : (-9886.27090624136, -5826.612192839276, 7718.076283243208)
  Point 2                : (-9556.480531049332, -5564.376458694018, 7480.636200018816)
  Distance               : 483.6393955765048
  Number of Euclidian Compute : 112
  Time Execution         : 1.508 mili seconds

  Brute Force
  Point 1                : (-9886.27090624136, -5826.612192839276, 7718.076283243208)
  Point 2                : (-9556.480531049332, -5564.376458694018, 7480.636200018816)
  Distance               : 483.6393955765048
  Number of Euclidian Compute : 8128
  Time Execution         : 18.907 mili seconds

  Want to visualized?
  Yes : 1
  No  : 0
  >>> 1
  
```

d. Jumlah titik 1000



```


Closest Pair  
of Points


  Input Terminal      : 1
  Input file         : 2
  >>> 1

  Number of Dimensions > 0 : 3
  Number of Points > 1    : 1000

  Divide and Conquer
  Point 1                : (-3667.9452648614515, 827.3924716223573, 8724.777511361677)
  Point 2                : (-3463.5849952659537, 897.7216952809795, 8768.853274012137)
  Distance               : 220.57196635744248
  Number of Euclidian Compute : 962
  Time Execution         : 6.901 mili seconds

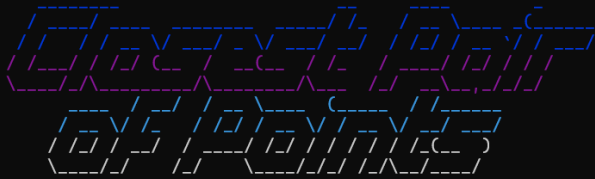
  Brute Force
  Point 1                : (-3463.5849952659537, 897.7216952809795, 8768.853274012137)
  Point 2                : (-3667.9452648614515, 827.3924716223573, 8724.777511361677)
  Distance               : 220.57196635744248
  Number of Euclidian Compute : 499500
  Time Execution         : 804.556 mili seconds

  Want to visualized?
  Yes : 1
  No  : 0
  >>> 1
  
```

## 2. Saat bidang n dimensi

### a. Jumlah titik 16 pada bidang 4 dimensi

```


Input Terminal      : 1
Input file          : 2
>>> 1

Number of Dimensions > 0 : 4
Number of Points > 1    : 16

Divide and Conquer
Point 1                : (-293.3669551859839, 4433.702431925642, -3351.3500899215587, 5312.155429476834)
Point 2                : (458.9507032906058, 3638.489749753706, -3672.390173943827, 4846.485033982759)
Distance               : 1232.1772282960824
Number of Euclidian Compute : 15
Time Execution         : 327.400 micro seconds

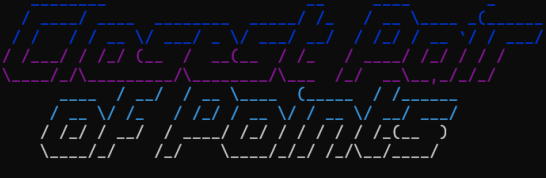
Brute Force
Point 1                : (458.9507032906058, 3638.489749753706, -3672.390173943827, 4846.485033982759)
Point 2                : (-293.3669551859839, 4433.702431925642, -3351.3500899215587, 5312.155429476834)
Distance               : 1232.1772282960824
Number of Euclidian Compute : 120
Time Execution         : 0.529 mili seconds

Exiting...

```

### b. Jumlah titik 64 pada bidang 5 dimensi

```


Input Terminal      : 1
Input file          : 2
>>> 1

Number of Dimensions > 0 : 5
Number of Points > 1    : 64

Divide and Conquer
Point 1                : (1179.929072416131, 6447.059807962702, 5557.109842505821, 7256.3710677886775, -3884.5363842627294)
Point 2                : (227.26032278152707, 7006.721246574307, 6009.225277321695, 7382.285865846094, -2178.6094785022024)
Distance               : 2085.964568962149
Number of Euclidian Compute : 78
Time Execution         : 2.615 mili seconds

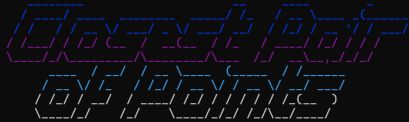
Brute Force
Point 1                : (227.26032278152707, 7006.721246574307, 6009.225277321695, 7382.285865846094, -2178.6094785022024)
Point 2                : (1179.929072416131, 6447.059807962702, 5557.109842505821, 7256.3710677886775, -3884.5363842627294)
Distance               : 2085.964568962149
Number of Euclidian Compute : 2016
Time Execution         : 14.057 mili seconds

Exiting...

```

### c. Jumlah titik 128 pada bidang 7 dimensi

```



```

Input Terminal      : 1
Input file         : 2
>>> 1

Number of Dimensions > 0 : 7
Number of Points > 1    : 128

Divide and Conquer
Point 1              : (-6996.431831108638, 4802.79234594956, 6909.212696416078, -808.6565176840595, -4651.0181164838405, 9607.355250847879, -8438.265374673567)
Point 2              : (-7644.24004145704, 3299.0960233484348, 8640.887315441796, -1038.2561678582788, -3685.1595830237984, 6382.741301558686, -8712.351392545175)
Distance             : 4139.844449654804
Number of Euclidian Compute : 274
Time Execution       : 12.323 mili seconds

Brute Force
Point 1              : (-7644.24004145704, 3299.0960233484348, 8640.887315441796, -1038.2561678582788, -3685.1595830237984, 6382.741301558686, -8712.351392545175)
Point 2              : (-6996.431831108638, 4802.79234594956, 6909.212696416078, -808.6565176840595, -4651.0181164838405, 9607.355250847879, -8438.265374673567)
Distance             : 4139.844449654804
Number of Euclidian Compute : 8128
Time Execution       : 66.770 mili seconds


Exiting...

```


```

d. Jumlah titik 1000 pada bidang 10 dimensi

```



```

Input Terminal      : 1
Input file         : 2
>>> 1

Number of Dimensions > 0 : 10
Number of Points > 1    : 1000

Divide and Conquer
Point 1              : (8328.315340515292, -5424.572912039427, -1215.8302008049686, -4757.020763956814, -2607.7400717359196, 1887.850495486251, 1098.903354128348, -2382.812082376633, -3066.8934211064243, -4972.046127890346)
Point 2              : (8202.680655654032, -5276.553745252714, 134.60005230677052, -3122.252834355235, -5294.227210002687, 1027.6516070548205, -1274.4062893143146, -4403.30783378436, -5562.781375017436, -4081.6050105332406)
Distance             : 5406.32230622204
Number of Euclidian Compute : 5132
Time Execution       : 826.238 mili seconds

Brute Force
Point 1              : (8328.315340515292, -5424.572912039427, -1215.8302008049686, -4757.020763956814, -2607.7400717359196, 1887.850495486251, 1098.903354128348, -2382.812082376633, -3066.8934211064243, -4972.046127890346)
Point 2              : (8202.680655654032, -5276.553745252714, 134.60005230677052, -3122.252834355235, -5294.227210002687, 1027.6516070548205, -1274.4062893143146, -4403.30783378436, -5562.781375017436, -4081.6050105332406)
Distance             : 5406.32230622204
Number of Euclidian Compute : 499500
Time Execution       : 4619.097 mili seconds

Exiting...

```


```

3. Saat inputan tidak sesuai

```






```

Input Terminal      : 1
Input file         : 2
>>> 1

Number of Dimensions > 0 : 0
Number of Points > 1    : 1

Invalid input. Exiting...

```


```

**BAB V**  
**TABLE**

Poin	YA	TIDAK
1. Program berhasil dikompilasi tanpa kesalahan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2. Program berhasil running	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3. Program dapat menerima masukan dan dan menuliskan luaran.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4. Luaran program sudah benar (solusi closest pair benar)	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5. Bonus 1 dikerjakan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6. Bonus 2 dikerjakan	<input checked="" type="checkbox"/>	<input type="checkbox"/>

## **Lampiran**

Link Repository Github: <https://github.com/tarsn/Tucil2-Stima>