

LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA

**Penyelesaian *The Closest Pair Problem* dengan
Algoritma *Divide and Conquer***



Disusun oleh:

Daniel Egiant Sitanggang	13521056
Rinaldy Adin	13521134

Program Studi Teknik
Informatika Sekolah Teknik
Elektro dan Informatika
Institut Teknologi Bandung
2023

I. Penjelasan Algoritma *Divide and Conquer*

Algoritma divide and conquer adalah suatu pendekatan dalam desain algoritma yang menguraikan masalah yang kompleks menjadi masalah yang lebih kecil dan lebih mudah dipecahkan. Pendekatan ini terdiri dari tiga tahap yaitu:

1. Divide
Masalah utama dibagi menjadi beberapa upa-masalah yang lebih kecil dan lebih sederhana namun masih memiliki kemiripan dengan masalah semula.
2. Conquer
Setiap upa-masalah dipecahkan secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar.
3. Combine
Solusi dari setiap upa-masalah digabungkan kembali menjadi solusi dari masalah semula

Dalam algoritma *divide and conquer*, proses pembagian masalah dilakukan berulang-ulang hingga diperoleh upa-masalah yang cukup sederhana dan mudah untuk dipecahkan secara langsung. Setelah upa-masalah berhasil dipecahkan, solusi dari setiap upa-masalah digabungkan kembali untuk membentuk solusi dari masalah utama.

Berikut adalah pseudocode dari algoritma divide and conquer.

```
procedure DIVIDEandCONQUER(input  $P$  : problem,  $n$  : integer)  
{ Menyelesaikan persoalan dengan algoritma divide and conquer  
  Masukan: masukan yang berukuran  $n$   
  Luaran: solusi dari persoalan semula  
}  
Deklarasi  
   $r$  : integer  
  
Algoritma  
  if  $n \leq n_0$  then {ukuran persoalan sudah cukup kecil}  
    SOLVE persoalan  $P$  yang berukuran  $n$  ini  
  else  
    DIVIDE menjadi 2 upa-persoalan,  $P_1$  dan  $P_2$ , masing-masing berukuran  $n/2$   
    DIVIDEandCONQUER( $P_1$ ,  $n/2$ )  
    DIVIDEandCONQUER( $P_2$ ,  $n/2$ )  
    COMBINE solusi dari  $P_1$  dan  $P_2$   
  endif
```

Gambar 1 Pseudocode Algoritma Divide and Conquer

Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf)

II. Deskripsi Algoritma Yang Digunakan

A. Masalah Pasangan Titik Terdekat

Dalam versinya yang sederhana, yaitu dalam bidang 2 dimensi, permasalahan pasangan titik terdekat adalah sebagai berikut, “Diberikan himpunan titik, P , yang terdiri dari n buah titik pada bidang 2 dimensi, (x_i, y_i) , $i = 1, 2, \dots, n$. Tentukan sepasang titik di dalam P yang jaraknya terdekat satu sama lain.” Dalam tugas kecil ini, persoalan tersebut dikembangkan menjadi pencarian titik terdekat dalam 3 dimensi, serta dalam dimensi yang lebih tinggi. Untuk mencari jarak d dari dua buah titik p dan q dalam dimensi n , dapat digunakan persamaan berikut,

$$d(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

Dengan pendekatan naif, kita dapat mencari pasangan titik dengan jarak paling kecil dengan memeriksa semua kemungkinan pasangan titik yang ada. Algoritma dengan pendekatan berikut disebut dengan algoritma *brute force*. Selain dengan algoritma *brute force*, persoalan ini dapat diselesaikan dengan pendekatan algoritma *divide and conquer*. Pendekatan algoritma ini lebih efisien daripada algoritma *brute force* karena algoritma ini mengandalkan geometri untuk memilih pasangan algoritma yang lebih sedikit untuk diperiksa dibandingkan dengan algoritma *brute force*. Program kami akan menunjukkan perbandingan dari kedua pendekatan tersebut dari jumlah pemeriksaan jarak titik dan dari waktu jalannya algoritma.

B. Algoritma Penyelesaian

Dalam menyelesaikan persoalan tersebut menggunakan pendekatan *divide and conquer* alur algoritma kami adalah sebagai berikut,

1. Urutkan titik yang ada berdasarkan salah satu sumbu, yang akan terjadi dalam **$O(n \log n)$** . Program kami memilih sumbu “paling tinggi” dalam dimensi tertentu, contohnya dalam dimensi 3, titik akan diurutkan berdasarkan sumbu- y .
2. Bagi himpunan titik yang sudah diurutkan tersebut menjadi dua himpunan **L** dan **R** secara sama banyak (kecuali jika terdapat titik berjumlah ganjil) berdasarkan urutannya dalam himpunan yang sudah terurut.
3. Cari pasangan titik dengan jarak terkecil δ dari kedua himpunan tersebut secara rekursif dengan kembali ke langkah pertama untuk masing-masing himpunan **L** dan **R**. Apabila masing-masing himpunan hanya tersisa 2 atau 3 titik, cari pasangan titik terdekat dengan memeriksa semua kemungkinan pasangan dalam himpunan tersebut.
4. Untuk mencari kemungkinan pasangan titik terdekat yang keduanya berada di himpunan yang berbeda, pilih titik-titik yang berjarak paling besar δ secara sumbu yang digunakan dalam pengurutan tadi dari garis/bidang median **M**, yaitu garis/bidang pembagi antara titik-titik di himpunan **L** dengan himpunan **R**. Masukkan titik tersebut ke dalam himpunan **S**.

5. Periksa kemungkinan titik terdekat dalam himpunan S dengan kembali menjalankan algoritma *divide and conquer* terhadap proyeksi titik dalam bidang/garis M , yaitu himpunan titik S' . Algoritma *divide and conquer* yang dijalankan berdasarkan proyeksi titik tersebut akan tetap dihitung jaraknya dalam dimensi paling tinggi meskipun algoritma akan memilih titik seperti halnya sedang dalam algoritma yang lebih rendah. Apabila himpunan titik S' berada pada ruang 1 dimensi, lanjutkan algoritma ke langkah berikut.
6. Urutkan titik pada S' berdasarkan sumbu paling rendah, yaitu sumbu- x , pengurutan terjadi dalam $O(n \log n)$.
7. Untuk setiap titik dalam S' , cari jarak terpendek dengan memeriksa titik dengan titik-titik berikutnya dalam pengurutan hingga sampai titik yang jarak dalam sumbu- x -nya lebih besar dari δ . Berdasarkan dimensi tertinggi dalam persoalan, jumlah titik lain yang akan diperiksa untuk setiap titik dalam S akan konstan sehingga pencarian pasangan titik terdekat dalam himpunan S' memiliki kompleksitas $O(n)$.
8. Bandingkan pasangan titik terdekat yang terdapat dalam L , R , dan S . Visualisasikan pasangan titik terdekat yang ditemukan.

III. Source Code Program dalam Bahasa Python

Berikut isi file dari program kami,

brute_force.py

```
from collections.abc import Callable

def closest_pair_brute(
    points: list[tuple[int]],
    getEuclideanDistance: Callable[[tuple[int], tuple[int]], float],
):
    if len(points) <= 1:
        return None, None

    min_points = []
    min = None

    for i in range(len(points)):
        for j in range(i + 1, len(points), 1):
            point1 = points[i]
            point2 = points[j]
            temp = getEuclideanDistance(point1, point2)
            if min == None or temp < min:
                min = temp
                min_points = [(point1, point2)]
            elif min != None and temp == min:
                min_points.append((point1, point2))

    return min_points, min
```

closest_pair.py

```
from divide_conquer import closest_pair
```

```

from brute_force import closest_pair_brute
from utils import generate_dots, DistFuncCounter
import timeit

def testing(iter=100):
    dim = 3
    points = generate_dots(dim=dim, num=1000)

    bf_counter = DistFuncCounter()
    bf_start = timeit.default_timer()
    bf = closest_pair_brute(points, bf_counter.getEuclideanDistance)
    bf_stop = timeit.default_timer()

    dnc_counter = DistFuncCounter()
    dnc_start = timeit.default_timer()
    dnc = closest_pair(points, dim, dnc_counter.getEuclideanDistance)
    dnc_stop = timeit.default_timer()

    if bf[2] != dnc[2]:
        print("incorrect")

    print(f"dnc time: {dnc_stop - dnc_start}")
    print(f"dnc count: {dnc_counter.count}")
    print(f"bf time: {bf_stop - bf_start}")
    print(f"bf count: {bf_counter.count}")

if __name__ == "__main__":
    testing(100)

```

divide_conquer.py

```

from collections.abc import Callable
from sorting import quicksort
from utils import appendPairsIfNotIn

def closest_pair_strip(
    points: list[tuple[int]],
    delta: float,
    dimension: int,
    getEuclideanDistance: Callable[[tuple[int], tuple[int]], float],
) -> tuple[list[tuple[tuple, tuple]], float]:
    midIdx: int = len(points) // 2
    median: float = (
        points[midIdx - 1][dimension - 1] + points[midIdx][dimension - 1]
    ) / 2
    stripPoints: list[tuple[int]] = [
        point for point in points if abs(median - point[dimension - 1]) <
delta
    ]

    if dimension == 2:
        quicksort(stripPoints, 0, len(stripPoints) - 1, 0)

        min_strip_points = []
        stripDelta: float = None

        for i in range(len(stripPoints)):
            for j in range(i + 1, len(stripPoints)):
                if stripPoints[j][0] - stripPoints[i][0] >= delta:
                    break

                dist = getEuclideanDistance(stripPoints[i],
stripPoints[j])
                if stripDelta == None or dist < stripDelta:

```

```

        min_strip_points = [(stripPoints[i],
stripPoints[j])]

        stripDelta = dist
        elif stripDelta != None and dist == stripDelta:
            min_strip_points.append((stripPoints[i],
stripPoints[j]))

    if stripDelta != None:
        return min_strip_points, stripDelta
    else:
        return None, None
else:
    return closest_pair(stripPoints, dimension - 1,
getEuclideanDistance)

def closest_pair(
    points: list[tuple[int]],
    dimension: int,
    getEuclideanDistance: Callable[[tuple[int], tuple[int]], float],
) -> tuple[list[tuple[tuple, tuple]], float]:
    if len(points) == 1:
        return None, None
    if len(points) <= 3:
        min_points = []
        delta = None
        for i in range(len(points)):
            for j in range(i + 1, len(points)):
                dist = getEuclideanDistance(points[i], points[j])
                if delta == None or dist < delta:
                    min_points = [(points[i], points[j])]
                    delta = dist
                elif delta != None and dist == delta:

```



```

        min_points.append((points[i], points[j]))

    return min_points, delta

quicksort(points, 0, len(points) - 1, dimension - 1)

midIdx: int = len(points) // 2
leftPoints: list[tuple[int]] = points[:midIdx]
rightPoints: list[tuple[int]] = points[midIdx:]

left_points, leftDelta = closest_pair(leftPoints, dimension,
getEuclideanDistance)
right_points, rightDelta = closest_pair(
    rightPoints, dimension, getEuclideanDistance
)

delta = min(leftDelta, rightDelta)

strip_points, stripDelta = closest_pair_strip(
    points, delta, dimension, getEuclideanDistance
)

if stripDelta != None:
    delta = min(delta, stripDelta)

min_points = []
if delta == leftDelta:
    min_points.extend(left_points)
if delta == rightDelta:
    min_points.extend(right_points)
if stripDelta != None and delta == stripDelta:
    appendPairsIfNotIn(min_points, strip_points)

```

```
return min_points, delta
```

IO.py

```
def read_from_file(path):
    points = []
    with open(path, "r") as file:
        for line in file:
            row = line.strip().split()
            row = tuple([int(num) for num in row])
            points.append(row)
    return points
```

main.py

```
from brute_force import *
from closest_pair import *
from divide_conquer import *
from utils import *
from IO import *

import matplotlib.pyplot as plt
import platform

def welcome_text():
    print(
        """
        -----
        /  ----|| |          | |   |  -- \          (-)
        | |      | |  ---   ---   ---   --- | | _ | | --) |__ _ - - - --
        """
    )
```

```

| |      | | / _ \ / __| / _ \ / __|| | __| | | ___// _ \` || || |'__|
| |____ | || (-) |\__ \| __/\__ \| | | | | | (-| || || |
\_____||_| \___/ |___/ \___||___/ \__| | | | \__,_|_|_|_|

"""

)

def menu_text():

print("=====MENU=====
==")

    print()
    print("1. Generate random          ")
    print("2. Import from file          ")
    print("3. Exit                      ")
    print()

print("=====
==")

def space(n=3):
    for _ in range(n):
        print()

def visualize(points, solution):
    solution_points = []
    for pair in solution:
        for point in pair:
            if point not in solution_points:
                solution_points.append(point)

```

```

fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
for point in points:
    if point not in solution_points:
        ax.scatter(point[0], point[1], point[2], c="b", marker="o")

for point in solution_points:
    ax.scatter(point[0], point[1], point[2], c="r", marker="o")

for point1, point2 in solution:
    ax.plot(
        [point1[0], point2[0]],
        [point1[1], point2[1]],
        [point1[2], point2[2]],
        c="k",
    )

ax.set_xlabel("x axis")
ax.set_ylabel("y axis")
ax.set_zlabel("z axis")

plt.show()

def compute(points):
    space(1)

    bf_counter = DistFuncCounter()
    bf_start = timeit.default_timer()
    bf_points, bf_dist = closest_pair_brute(points,
bf_counter.getEuclideanDistance)
    bf_stop = timeit.default_timer()

```

```

    dim = len(points[0])
    dnc_counter = DistFuncCounter()
    dnc_start = timeit.default_timer()
    dnc_points, dnc_dist = closest_pair(points, dim,
dnc_counter.getEuclideanDistance)
    dnc_stop = timeit.default_timer()

print("=====SOLUTION=====
===")
    print()
    if len(dnc_points) == 1:
        print(
            f"Closest pair                :
{dnc_points[0][0]}, {dnc_points[0][1]}"
        )
    else:
        print(f"Closest pairs                :")
        for point1, point2 in dnc_points:
            print(f"{point1}, {point2}")
        print(f"Closest distance                : {dnc_dist}")
        print()

print("=====STATISTICS=====
===")
    print(f"DNQ execution time                : {(dnc_stop -
dnc_start)*1000} ms")
    print(f"DNQ distance computation count        :
{dnc_counter.count}")
    print(f"Brute-force execution time                : {(bf_stop -
bf_start)*1000} ms")
    print(f"Brute-forcedistance computation count :

```

```

{bf_counter.count}")
    print()
    print(f"Running on: {platform.processor()}")

    if dim == 3:
        cc = str(input("Do you want to visualize (y/n)? "))
        if cc == "y":
            visualize(points, dnc_points)

def main():
    welcome_text()
    while True:
        menu_text()
        choice = int(input("> "))

        if choice == 1:
            num = int(input("Enter the number of points (n >= 2) : 
"))

            if num <= 1:
                print("Please enter a number >= 2")
                continue

            dim = int(input("Enter the number of dimensions (d>=1) : 
"))

            if dim < 1:
                print("Please enter a number >= 1")
                continue

            x = int(input("Enter the range of randomized value : "))
            points = generate_dots(num, x, dim)

```

```

        compute(points)

    elif choice == 2:
        path = str(input("Enter the txt file path
: "))

        try:
            points = read_from_file(path)
            print(points)
            compute(points)
        except:
            print("File not found or out of format")

    elif choice == 3:
        print("Bye-bye~")
        exit(0)

    else:
        print("Masukan salah, harap masukkan (1-3)")

if __name__ == "__main__":
    main()

```

sorting.py

```

def partition(vectors, low, high, keyIdx):
    pivot = vectors[high][keyIdx]
    i = low - 1

    for j in range(low, high):
        if vectors[j][keyIdx] <= pivot:
            i = i + 1

```

```

        (vectors[i], vectors[j]) = (
            vectors[j],
            vectors[i],
        )

    (vectors[i + 1], vectors[high]) = (
        vectors[high],
        vectors[i + 1],
    )
    return i + 1

def quicksort(vectors, low, high, keyIdx=0):
    if low < high:
        pi = partition(vectors, low, high, keyIdx)

        quicksort(vectors, low, pi - 1, keyIdx)
        quicksort(vectors, pi + 1, high, keyIdx)

```

utils.py

```

import math
from random import randint

def generate_dots(num=100, x=100, dim=3):
    return [[randint(-x, x) for _i in range(dim)] for _j in range(num)]

def distance(p1, p2):
    return math.sqrt(sum((p1[i] - p2[i]) ** 2 for i in range(len(p1))))

```



```

def appendPairsIfNotIn(
    target: list[tuple[tuple, tuple]], pairs: list[tuple[tuple, tuple]]
):
    toAdd = []
    for pair in pairs:
        found = False
        for tgt in target:
            if (tgt[0] == pair[0] and tgt[1] == pair[1]) or (
                tgt[0] == pair[1] and tgt[1] == pair[0]
            ):
                found = True
        if not found:
            toAdd.append(pair)

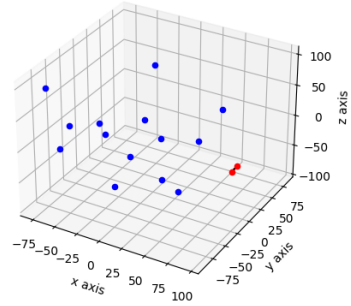
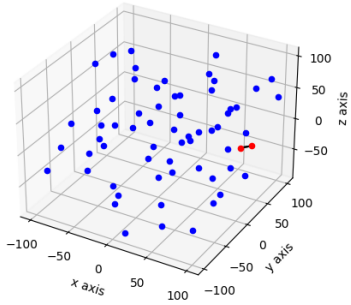
    target.extend(toAdd)

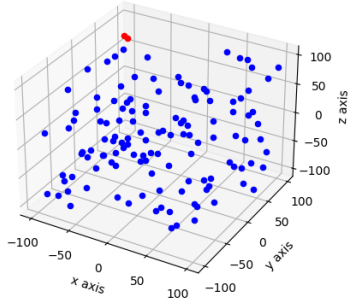
class DistFuncCounter:
    def __init__(self):
        self.count = 0

    def getEuclideanDistance(self, p1, p2):
        self.count += 1
        return distance(p1, p2)

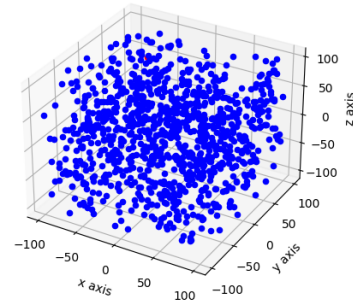
```

IV. Masukan dan Keluaran Program

Masukan	Luaran
<p>Number of points (n) : 16 Number of dimensions (d) : 3 Range of randomized value : 100</p>	 <pre> Enter the number of points (n >= 2) : 16 Enter the number of dimensions (d>=1) : 3 Enter the range of randomized value : 100 =====SOLUTION===== Closest pair : (68, 49, -68), (71, 32, -63) Closest distance : 17.97220075561143 =====STATISTICS===== DNQ execution time : 0.16980001237243414 ms DNQ distance computation count : 45 Brute-force execution time : 0.21950004156678915 ms Brute-forcedistance computation count : 120 Running on: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD Do you want to visualize (y/n)? y </pre>
<p>Number of points (n) : 64 Number of dimensions (d) : 3 Range of randomized value : 100</p>	

	<pre>Enter the number of points (n >= 2) : 64 Enter the number of dimensions (d>=1) : 3 Enter the range of randomized value : 100 =====SOLUTION===== Closest pair : (85, 36, -9), (97, 40, -4) Closest distance : 13.601470508735444 =====STATISTICS===== DNQ execution time : 1.0175000643357635 ms DNQ distance computation count : 267 Brute-force execution time : 4.073300049640238 ms Brute-forcedistance computation count : 2016 Running on: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD Do you want to visualize (y/n)? y</pre>
Number of points (n) : 128 Number of dimensions (d) : 3 Range of randomized value : 100	 <pre>Enter the number of points (n >= 2) : 128 Enter the number of dimensions (d>=1) : 3 Enter the range of randomized value : 100 =====SOLUTION===== Closest pair : (-12, -26, 16), (-13, -23, 12) Closest distance : 5.0990195135927845 =====STATISTICS===== DNQ execution time : 2.125500002875924 ms DNQ distance computation count : 583 Brute-force execution time : 12.062699999660254 ms Brute-forcedistance computation count : 8128 Running on: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD Do you want to visualize (y/n)? y</pre>

Number of points (n) : 1000
 Number of dimensions (d) : 3
 Range of randomized value : 100



```

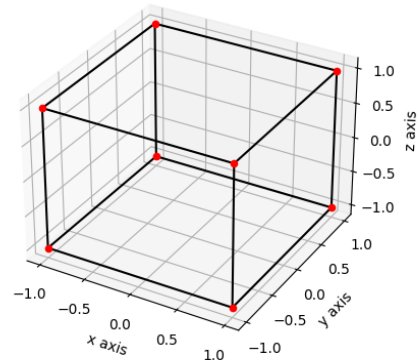
Enter the number of points (n >= 2) : 1000
Enter the number of dimensions (d>=1) : 3
Enter the range of randomized value : 100

=====SOLUTION=====
Closest pair : (-14, -75, -33), (-13, -76, -32)
Closest distance : 1.7320508075688772

=====STATISTICS=====
DNQ execution time : 22.93370000552386 ms
DNQ distance computation count : 6384
Brute-force execution time : 748.3196000102907 ms
Brute-forcedistance computation count : 499500

Running on: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
Do you want to visualize (y/n)? y
  
```

Number of points (n) : 8
 Number of dimensions (d) : 3
 Input File Contents:
 1 1 1
 -1 1 1
 1 -1 1
 1 1 -1
 -1 1 -1
 -1 -1 1
 1 -1 -1
 -1 -1 -1



```

> 2
Enter the txt file path : cube.txt
[[1, 1, 1], [-1, 1, 1], (1, -1, 1), (1, 1, -1), (-1, 1, -1), (-1, -1, 1), (1, -1, -1), (-1, -1, -1)]

=====SOLUTION=====

Closest pairs :
(1, 1, -1), (-1, 1, -1)
(1, -1, -1), (-1, -1, -1)
(-1, -1, -1), (-1, 1, -1)
(1, 1, -1), (1, -1, -1)
(-1, 1, 1), (-1, -1, 1)
(1, -1, 1), (1, 1, 1)
(-1, -1, 1), (1, -1, 1)
(-1, 1, 1), (1, 1, 1)
(-1, -1, -1), (-1, -1, 1)
(1, -1, -1), (1, -1, 1)
(-1, 1, 1), (-1, 1, -1)
(1, 1, -1), (1, 1, 1)
Closest distance : 2.0

=====STATISTICS=====
DMQ execution time : 0.24969992227852345 ms
DMQ distance computation count : 36
Brute-force execution time : 0.8754999928176403 ms
Brute-forcedistance computation count : 28

Running on: Intel64 Family 6 Model 142 Stepping 12, GenuineIntel
Do you want to visualize (y/n)? y

```

V. Referensi

Banik, A. (n.d.). *Closest Pair Problem*.

<http://www.niser.ac.in/~aritra/CG/ClosestPairProblem.pdf>

Indyk, P. (2003, October 30). *Closest Pair*.

<http://people.csail.mit.edu/indyk/6.838-old/handouts/lec17.pdf>

Munir, R. (2021a). *Algoritma Divide and Conquer (Bagian 1)*.

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf)

Munir, R. (2021b). *Algoritma Divide and Conquer (Bagian 2)*.

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf)

VI. Lampiran

Pranala Repository Github

https://github.com/egijago/Tucil2_13521056_13521134

Matriks Ketercapaian

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	