

**PENCARIAN PASANGAN TITIK TERDEKAT
MENGUNAKAN ALGORITMA
DIVIDE AND CONQUER**

**LAPORAN TUGAS KECIL
IF2211
STRATEGI ALGORITMA**

Oleh

**Varraz Hazzandra Abrar
13521020
K03**



**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
INSTITUT TEKNOLOGI BANDUNG
2022**

DESKRIPSI TEORI

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 antara dua titik dihitung dengan rumus Euclidian.

DASAR TEORI

1. Divide and Conquer

Algoritma divide and conquer merupakan teknik menyelesaikan suatu persoalan dengan memecah persoalan menjadi beberapa upa-persoalan yang lebih kecil sampai upa-persoalan tidak bisa dipecah lagi. Algoritma divide and conquer terbagi menjadi 3 langkah, yaitu divide, conquer, dan combine.

1. Langkah divide : memecah persoalan menjadi beberapa upa-persoalan yang punya kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya berukuran hampir sama).
2. Langkah conquer (solve) menyelesaikan masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar).
3. Langkah combine menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

2. Penerapan Divide and Conquer pada Pencarian *Closest Pair* di Ruang 3 dimensi

Pertama, titik-titik akan diurutkan berdasarkan koordinat x, y, dan z secara terpisah tiap koordinatnya. Titik-titik yang sudah terurut akan dimasukkan ke dalam *list* untuk masing-masing koordinat. Lalu, akan dilakukan pemanggilan fungsi yang menerapkan algoritma *divide and conquer*.

Algoritma *divide and conquer* akan diterapkan dengan membuat fungsi sendiri. Pertama, akan dicek banyak titik-titik pada *list* untuk koordinat x. Jika banyak titik ≤ 3 , algoritma *brute force* akan diterapkan pada titik-titik tersebut untuk mencari *closest pair*. Jika lebih dari 3, langkah pertama adalah membagi titik-titik pada *list* untuk koordinat x menjadi 2 bagian, yaitu bagian kiri dan kanan dengan indeks tengah berupa *mid*. Fungsi akan memanggil dirinya sendiri untuk mencari *closest pair* di kedua bagian. *Closest pair* keduanya akan dibandingkan dan didapatkan *closest pair* terkecil di antara kedua sisi dan disimpan di variable *minDist*. Lalu, fungsi akan memeriksa area sekitar *mid* dengan jangkauan sejauh *minDist* ke kanan dan ke kiri dari indeks *mid*. Titik-titik yang berada di dalam area jangkauan akan disimpan dalam *list strip*. Akan dicari *closest pair* di strip, lalu dibandingkan dengan *minDist* yang menyimpan *closest pair* hasil perbandingan bagian kiri dan kanan. Hasil perbandingan terbaru akan disimpan di variable *minDist*. Akan didapat berupa jarak terdekat dan *closest pair*

IMPLEMENTASI

Penulis membuat dua *file* untuk menyelesaikan persoalan *closest pair* di ruang 3 dimensi.

1. algoClPair.py

Program ini berisi penerapan algoritma *brute force* dan *divide and conquer* dalam menemukan *closest pair*. Program ini terdiri dari beberapa fungsi

a. fungsi `eucDist` : fungsi ini mencari jarak Euclidian antara dua titik.

```
def eucDist(p1, p2):  
    return ((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2 + (p1[2]-p2[2])**2)**0.5
```

b. fungsi `bfClosestPair` : fungsi ini mencari *closest pair* dengan algoritma brute force.

```
# Algoritma brute force  
def bfClosestPair(points):  
    minDist = float('inf')  
    clPair = None  
    countEuc = 0  
    totPoints = len(points)  
    for i in range(totPoints):  
        for j in range(i+1, totPoints):  
            dist = eucDist(points[i], points[j])  
            countEuc += 1  
            if dist < minDist:  
                clPair = (points[i], points[j])  
                minDist = dist  
    return clPair, minDist, countEuc
```

c. fungsi `sorting` : fungsi ini mengurutkan titik-titik berdasarkan koordinat x, y, dan z.

```
def sorting(points):  
    pointsX = sorted(points, key=lambda x: x[0])  
    pointsY = sorted(points, key=lambda x: x[1])  
    pointsZ = sorted(points, key=lambda x: x[2])  
    return pointsX, pointsY, pointsZ
```

d. fungsi `dcClosestPair` : fungsi ini mencari *closest pair* dengan algoritma *divide and conquer*.

```
def dcClosestPair(pointsX, pointsY, pointsZ):
    totPointsX = len(pointsX)
    if totPointsX <= 3:
        return bfClosestPair(pointsX)
    else:
        mid = totPointsX // 2
        midPoint = pointsX[mid]
        leftPointsY = [p for p in pointsY if p[0] <= midPoint[0]]
        rightPointsY = [p for p in pointsY if p[0] > midPoint[0]]
        leftPointsZ = [p for p in pointsZ if p[0] <= midPoint[0]]
        rightPointsZ = [p for p in pointsZ if p[0] > midPoint[0]]
        leftPair, leftMinDist, leftCountEuc = dcClosestPair(pointsX[:mid],
leftPointsY, leftPointsZ)
        rightPair, rightMinDist, rightCountEuc = dcClosestPair(pointsX[mid:],
rightPointsY, rightPointsZ)
        if leftMinDist < rightMinDist:
            minDist = leftMinDist
            clPair = leftPair
        else:
            minDist = rightMinDist
            clPair = rightPair
        strip = [p for p in pointsY if abs(p[0] - midPoint[0]) < minDist]
        totPointsStrip = len(strip)
        for i in range(totPointsStrip):
            j = i + 1
            while j < totPointsStrip and strip[j][1] - strip[i][1] < minDist:
                dist = eucDist(strip[i], strip[j])
                if dist < minDist:
                    minDist = dist
                    clPair = (strip[i], strip[j])
                j += 1
        countEuc = leftCountEuc + rightCountEuc + j - i
        return clPair, minDist, countEuc
```

2. eksekutorAlgoClPair.py

Program ini merupakan program utama yang akan mengeluarkan keluaran hasil pencarian *closest pair*.

a. *import library* dan *file* `algoClPair.py` untuk memanggil fungsi di dalamnya

```
import random
import time
import algoClPair
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

b. Pemrosesan masukan titik-titik secara acak sejumlah n dan visualisasi titik-titik

```
n = int(input("Masukkan jumlah titik : "))

points = [(random.randint(0, 2000), random.randint(0, 2000), random.randint(0, 2000)) for i in range(n)]
print("Titik-titik yang dibangkitkan : ")

for p in points:
    print(p)

visual = plt.figure()
ax = visual.add_subplot(111, projection='3d')
xs = [p[0] for p in points]
ys = [p[1] for p in points]
zs = [p[2] for p in points]
ax.scatter(xs, ys, zs, c='black')
```

c. Pemrosesan pencarian *closest pair* dengan algoritma *brute force*

```
# Menggunakan algoritma brute force
starting = time.time()
clPairBF, minDistBF, countEucBF = algoClPair.bfClosestPair(points)
ending = time.time()
durationBF = ending - starting
```

d. Pemrosesan pencarian *closest pair* dengan algoritma *divide and conquer*

```
# Menggunakan algoritma divide and conquer
starting = time.time()
pointsX, pointsY, pointsZ = algoClPair.sorting(points)
clPairDC, minDistDC, countEucDC = algoClPair.dcClosestPair(pointsX, pointsY, pointsZ)
ending = time.time()
durationDC = ending - starting
```

e. Pemberian warna pada titik-titik dan penanda pada *closest pair*

```
# Mewarnai titik-titik pada pasangan terdekat dengan warna kuning dan garis penghubung antara pasangan terdekat deng
for p in clPairDC:
    ax.scatter(p[0], p[1], p[2], c='yellow')
    x = [p[0], clPairDC[0][0]]
    y = [p[1], clPairDC[0][1]]
    z = [p[2], clPairDC[0][2]]
    ax.plot(x, y, z, c='red')
```

f. Mencetak keluaran dari program dan pemanggilan fungsi yang menampilkan visualisasi hasil program

```
print("\nHasil pencarian pasangan titik terdekat dengan algoritma brute force : ")
print("Pasangan titik terdekat :", clPairBF)
print("Jarak terdekat:", minDistBF)
print("Jumlah operasi euclidean terpakai:", countEucBF)
print("Lama waktu pencarian:", durationBF, "detik")

print("\nHasil pencarian pasangan titik terdekat dengan algoritma divide and conquer:")
print("Pasangan titik terdekat:", clPairDC)
print("Jarak terdekat:", minDistDC)
print("Jumlah operasi euclidean terpakai:", countEucDC)
print("Lama waktu pencarian:", durationDC, "detik")

plt.show()
```

EKSPERIMEN

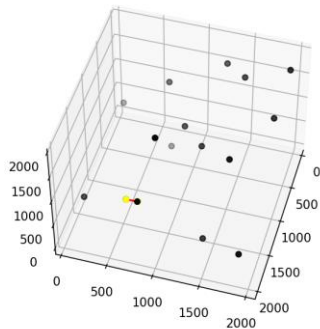
Eksperimen dilakukan di laptop Acer Swift SF314-510G

$n = 16$

```
PS D:\koding> python3 eksekutorAlgoClPair.py
Masukkan jumlah titik : 16
Titik-titik yang dibangkitkan :
(251, 1963, 1933)
(615, 1892, 1418)
(689, 952, 986)
(230, 1480, 1581)
(1391, 835, 1663)
(95, 78, 346)
(1648, 1415, 168)
(222, 647, 1164)
(1886, 787, 1045)
(1588, 104, 413)
```

```
Hasil pencarian pasangan titik terdekat dengan algoritma brute force :
Pasangan titik terdekat : ((1886, 787, 1045), (1966, 695, 1172))
Jarak terdekat: 176.04828883008207
Jumlah operasi euclidean terpakai: 120
Lama waktu pencarian: 0.0 detik
```

```
Hasil pencarian pasangan titik terdekat dengan algoritma divide and conquer:
Pasangan titik terdekat: ((1886, 787, 1045), (1966, 695, 1172))
Jarak terdekat: 176.04828883008207
Jumlah operasi euclidean terpakai: 15
Lama waktu pencarian: 0.0 detik
```



$n = 64$


```

PS D:\koding> python3 eksekutorAlgoClPair.py
Masukkan jumlah titik : 64
Titik-titik yang dibangkitkan :
(211, 1123, 1377)
(1404, 1328, 730)
(361, 1497, 240)
(1118, 915, 1469)
(313, 1239, 1350)
(1166, 1661, 1357)
(495, 1960, 218)
(1098, 424, 1147)
(1619, 1987, 1051)
(664, 1018, 1694)

```

```

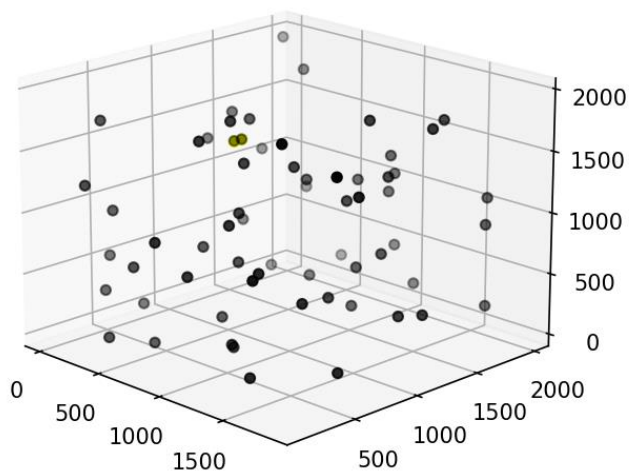
Hasil pencarian pasangan titik terdekat dengan algoritma brute force :
Pasangan titik terdekat : ((313, 1239, 1350), (383, 1231, 1387))
Jarak terdekat: 79.58014827832379
Jumlah operasi euclidean terpakai: 2016
Lama waktu pencarian: 0.004506111145019531 detik

```

```

Hasil pencarian pasangan titik terdekat dengan algoritma divide and conquer:
Pasangan titik terdekat: ((313, 1239, 1350), (383, 1231, 1387))
Jarak terdekat: 79.58014827832379
Jumlah operasi euclidean terpakai: 63
Lama waktu pencarian: 0.0 detik

```



$n = 128$

```

PS D:\koding> python3 eksekutorAlgoClPair.py
Masukkan jumlah titik : 128
Titik-titik yang dibangkitkan :
(1861, 926, 1310)
(1785, 82, 752)
(1254, 1952, 163)
(1505, 66, 689)

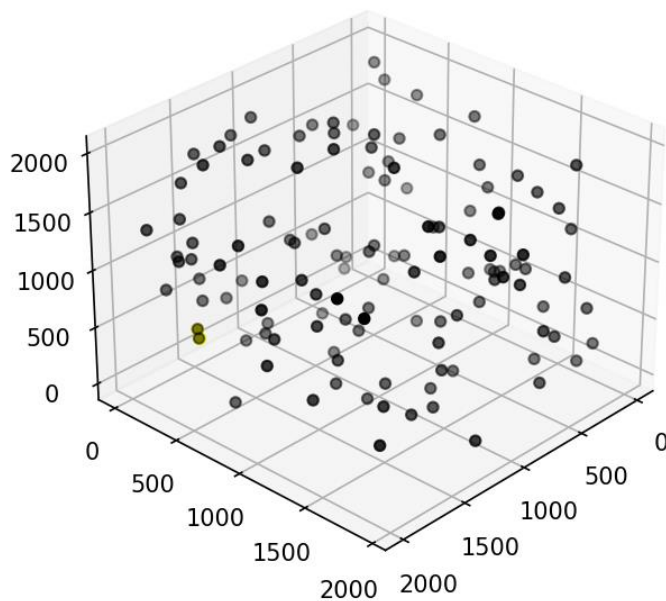
```

```

Hasil pencarian pasangan titik terdekat dengan algoritma brute force :
Pasangan titik terdekat : ((1609, 156, 263), (1592, 129, 322))
Jarak terdekat: 67.07458535093602
Jumlah operasi euclidean terpakai: 8128
Lama waktu pencarian: 0.011749505996704102 detik

Hasil pencarian pasangan titik terdekat dengan algoritma divide and conquer:
Pasangan titik terdekat: ((1592, 129, 322), (1609, 156, 263))
Jarak terdekat: 67.07458535093602
Jumlah operasi euclidean terpakai: 127
Lama waktu pencarian: 0.0 detik

```



n = 1000

```

PS D:\koding> python3 eksekutorAlgoClPair.py
Masukkan jumlah titik : 1000

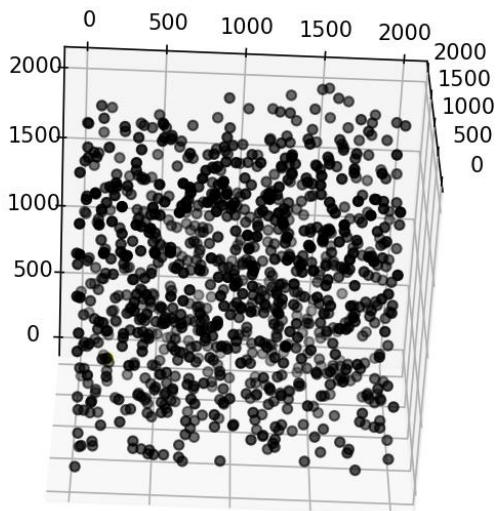
```

```

Hasil pencarian pasangan titik terdekat dengan algoritma brute force :
Pasangan titik terdekat : ((211, 190, 1373), (209, 184, 1372))
Jarak terdekat: 6.4031242374328485
Jumlah operasi euclidean terpakai: 499500
Lama waktu pencarian: 0.7513425350189209 detik

Hasil pencarian pasangan titik terdekat dengan algoritma divide and conquer:
Pasangan titik terdekat: ((209, 184, 1372), (211, 190, 1373))
Jarak terdekat: 6.4031242374328485
Jumlah operasi euclidean terpakai: 1023
Lama waktu pencarian: 0.011166572570800781 detik

```



LAMPIRAN

Pranala repositori : https://github.com/varrazha/Tucil2_13521020

| POIN | YA | TIDAK |
|--|----|-------|
| Program berhasil dikompilasi tanpa kesalahan | ✓ | |
| Program berhasil running | ✓ | |
| Program dapat menerima masukan dan memberikan luaran | ✓ | |
| Luaran program sudah benar (solusi closest pair benar) | ✓ | |
| Bonus 1 dikerjakan | ✓ | |
| Bonus 2 dikerjakan | | ✓ |