Tugas Kecil 2 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer

Disusun oleh:

Kenneth Dave Bahana 13521145
Yobel Dean Christopher 13521067

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

2023

Landasan Teori

1.1 Penjelasan Algoritma Divide and Conquer

Algoritma Divide and Conquer dari kata *Divide* yang berarti membagi dan *Conquer* yang berarti menaklukan, merupakan algoritma yang membagi suatu persoalan dengan membagi - bagi persoalan tersebut menjadi beberapa upa-persoalan yang memiliki kemiripan dalam ukuran yang lebih kecil, kemudian setiap upa-persoalan tersebut diselesaikan masalahnya (*Conquer*) dan setiap solusi dari upa-persoalan tersebut digabung kembali dalam tahap *Combine* hingga akhirnya dengan menyelesaikan setiap upa-persoalan tersebut dapat menyelesaikan persoalan utamanya.

Pendekatan dari algoritma *Divide and Conquer* ini merupakan sebuah strategi memperkecilkan masalah kemudian diselesaikan secara "*brute force*" untuk setiap upa-persoalan yang telah dibagi-bagikan yang akan digabungkan lagi untuk mendapatkan jarak euclidean terdekat global.

1.2 Penjelasan Algoritma Brute Force

Algoritma *brute force* merupakan algoritma dengan pendekatan yang lempang(*straightforward*) untuk memecahkan suatu persoalan. Algoritma ini bersifat luas karena dapat berupa berbagai algoritma yang sifatnya sederhana dan caranya bersifat jelas dipahami manusia karena ide pembentukan algoritma ini didasarkan apa yang terlintas dalam pemikiran ketika suatu program ingin dibuat.

Pendekatan utama dari algoritma *brute force* adalah dengan menguji seluruh kemungkinan dalam mendapatkan hasil yang memenuhi syarat sebagai solusi dari suatu permasalahan. Pendekatan ini yang menyebabkan algoritma *brute force* seringkali bersifat tidak efisien dikarenakan algoritmanya tidak mengutamakan efisiensi dan kecepatan eksekusi program, namun mengutamakan program dapat berjalan dengan cara yang secara *straightforward* dan optimal sehingga algoritma ini dapat juga disebut algoritma naif (naïve algorithm) dikarenakan menggunakan usaha untuk menguji segala kemungkinan dalam

pembentukkan algoritmanya, dibandingkan dengan memikirkan pengujian cara yang efisien atau membutuhkan jumlah tahapan yang lebih sedikit dalam algoritmanya sehingga tidak perlu menguji setiap kemungkinan.

1.3 Pengukuran Pasangan Titik Jarak Terdekat 3D

Dalam menyelesaikan persoalan sejumlah titik yang secara acak dimunculkan pada suatu bidang, Perhitungan dari jarak antar-titik itu sendiri menggunakan rumus jarak euclidean. Dalam dimensi tiga, perhitungan jarak antara dua titik dihitung dengan rumus berikut.

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

Perhitungan dan jumlah operasi dalam perhitungan jarak ini akan dipengaruhi dalam implementasi algoritma *divide* and *conquer* berdasarkan jumlah dimensi yang dipilih atau digenerasikan pada titik - titiknya. Untuk tiga dimensi ini, dilakukan perhitungan jarak selisih untuk setiap sumbunya, yaitu tiga penambahan operasi.

Implementasi

2.1 Algoritma Program

```
def quickSort(list, key):
    if len(list) <= 1:
        return list
    else:
        pivot = list.pop()

    greaterPivot = []
    lesserPivot = []
    for titik in list:
        if titik[key] > pivot[key]:
            greaterPivot.append(titik)
        else:
            lesserPivot.append(titik)

    return quickSort(lesserPivot, key) + [pivot] + quickSort(greaterPivot,
key)

def dist(p1, p2):
```

```
    global euclid_count
    euclid_count += 1
    distance = 0
    for i in range(len(p1)):
        distance += (p1[i] - p2[i])**2
    return math.sqrt(distance)

def bruteForce(P, n):
    min_dist = 999999
    for i in range(n):
        for j in range(i+1, n):
            if dist(P[i], P[j]) < min_dist:
                min_dist = dist(P[i], P[j])
                closest_pair = (P[i], P[j])
    return closest_pair

def min(x, y):
    if x < y:
        return x
    else:
        return y

def stripClosest(strip, size, d, closest_pair):
    min_dist = d
    #printPoints(strip)
    for i in range(size):
        for j in range(i+1, size):
            if (strip[j][0] - strip[i][0]) >= min_dist:
                break
            if dist(strip[i], strip[j]) < min_dist:
                min_dist = dist(strip[i], strip[j])
                closest_pair = (strip[i], strip[j])
    return closest_pair

def closestDots(P, n):
    if n <= 3:
        return bruteForce(P, n)
    mid = n//2
    midPoint = P[mid]
    left_pair = closestDots(P[:mid], mid)
```

```
    right_pair = closestDots(P[mid:], n - mid)
    left_closest = dist(*left_pair)
    right_closest = dist(*right_pair)
    d = min(left_closest, right_closest)
    if left_closest < right_closest:
        closest_pair = left_pair
    else:
        closest_pair = right_pair
    strip = []
    for i in range(n):
        if abs(P[i][0] - midPoint[0]) < d:
            strip.append(P[i])
    if len(strip) != 0:
        closest_pair_in_strip = stripClosest(strip, len(strip), d,
closest_pair)
        if dist(*closest_pair) < dist(*closest_pair_in_strip):
            return closest_pair
        else:
            return closest_pair_in_strip

    else:
        return closest_pair

def DivAndConq(P, n):
    P = quickSort(P, 0)
    return closestDots(P, n)

def createRandomPoints(n, dimension):
    points = []
    for i in range(n):
        p = []
        for j in range(dimension):
            p.append(round(random.uniform(1, 100), 2))
        points.append(p)
    return points

def printPoints(points):
    print("The randomize points are,")
    for point in points:
        print("(", end='')
```

```
        for i in range(len(points[0])):
            print(point[i], end= '')
            if i < len(points[0])-1:
                print(", ")
        print(")")

def printPointPair(pair):
    p1, p2 = pair
    print("Point 1: (", end='')
    for j in range(len(p1)):
        print (p1[j], end= '')
        if (j < len(p1)-1):
            print (', ', end ='')
    print(")")

    print("Point 2: (", end= '')
    for j in range(len(p2)):
        print (p2[j], end='')
        if (j < len(p2)-1):
            print (', ', end= '')
    print(")")


def visualizePoints(points, closest_pair=None):
    fig = plt.figure()
    if len(points[0]) == 3:
        ax = fig.add_subplot(111, projection='3d')
    elif len(points[0]) == 2:
        ax = fig.add_subplot(111)
    else:
        ax = fig.add_subplot(111)
    x = []
    for i in range(len(points[0])):
        x.append([point[i] for point in points])
    if closest_pair:
        p1, p2 = closest_pair
        if (len(p1) == 3):
            ax.scatter(p1[0], p1[1], p1[2], color='r', s=100)
            ax.scatter(p2[0], p2[1], p2[2], color='r', s=100)
        elif (len(p1) == 2):
```

```
            ax.scatter(p1[0], p1[1], color='r', s=100)
            ax.scatter(p2[0], p2[1], color='r', s=100)
        else:
            ax.plot(p1[0], 0.0, color='r')
            ax.plot(p2[0], 0.0, color='r')

    if (len(p1) == 3):
        ax.plot([p1[0], p2[0]], [p1[1], p2[1]], [p1[2], p2[2]], color =
'r', linewidth = 5)
        ax.scatter(x[0], x[1], x[2])
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')

    elif (len(p1) == 2):
        ax.plot([p1[0], p2[0]], [p1[1], p2[1]], color = 'r', linewidth =
5)
        ax.scatter(x[0], x[1])
        ax.set_xlabel('X')
        ax.set_ylabel('Y')

    else:
        y = [0 for i in range(len(x[0]))]
        ax.plot([p1[0], p2[0]], [0, 0], color = 'r', linewidth = 5)
        ax.scatter(x[0], y)
        ax.set_xlabel('X')
        ax.set_ylabel('Y')

    plt.show()

# Main
print()
print("Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and
Conquer")
print()

n_titik = int(input("Jumlah titik: "))
while n_titik <= 1:
    print("Jumlah titik untuk diuji tidak valid. Silahkan input ulang
dengan jumlah titik lebih dari 1!")
```

```
    n_titik = int(input("Jumlah titik: "))
dimens = int(input("Dimensi: "))
while dimens <= 0:
    print("Dimensi tersebut tidak valid. Silahkan input ulang dimensi
dengan nilai 1 atau ke atas.")
    dimens = int(input("Dimensi: "))
P = []
P = createRandomPoints(n_titik, dimens)
P2 = P.copy()
# printPoints(P)
print()


# BruteForce
start_bf = time.time()
closest_pair2 = bruteForce(P2, n_titik)
end_bf = time.time()
# Divide & Conquer
euclid_count_bf = euclid_count

start_time = time.time()
closest_pair = DivAndConq(P, n_titik)
end_time = time.time()
# printPoints(P2)



print()
print("Pasangan titik terdekat dengan algoritma Divide & Conquer:")
printPointPair(closest_pair)
print()
print("Pasangan titik terdekat dengan algoritma BruteForce:")
printPointPair(closest_pair2)
print()

print("Jarak pasangan titik terdekat Divide & Conquer: ",
dist(*closest_pair)) #round(dist(*closest_pair), 2))
print()
print("Jarak pasangan titik terdekat Brute Force: ", dist(*closest_pair2))
print()
```

```
print("Jumlah operasi euclidean algoritma Divide & Conquer: ",
euclid_count-euclid_count_bf)
print("Jumlah operasi euclidean algoritma Brute Force: ", euclid_count_bf)
print("Waktu Eksekusi Divide & Conquer ", 1000*(end_time - start_time),
"miliseconds")
print("Waktu Eksekusi Brute Force ", 1000*(end_bf - start_bf), "
miliseconds")
print()
if dimens <= 3 & dimens > 0:
        visualize = input("Apakah ingin ditampilkan visualisasinya? (Y /
N)")
        while visualize not in {'Y', 'N', 'y', 'n'}:
            print("Input tidak valid. Silahkan input ulang!")
            visualize = input("Apakah ingin ditampilkan
visualisasinya?(Y/N) ")
        if (visualize in {'Y', 'y'}):
            visualizePoints(P, closest_pair)
else:
    print("Maaf, tidak dapat divisualisasikan pada dimensi,", dimens)
```

2.2 Penjelasan Algoritma

Pertama, diimplementasikan fungsi *quicksort*. *Quicksort* yang diimplementasikan dimulai dengan memilih pivot pada akhir list dan di pop (dikeluarkan dari *list*) dan kemudian setiap nilai dalam list dikomparasikan dengan pivot. Apabila lebih besar, dimasukkan ke dalam list baru yang berisi nilai - nilai lebih besar dari pivot dan apabila lebih kecil atau sama dengan, masuk ke dalam list baru berisi nilai - nilai lebih kecil dari pivot. Kemudian kedua list tersebut secara rekursif akan dilakukan *quicksort* lagi serta nilai pivot ditambahkan di antara kedua *list* tersebut.

Dua fungsi utama yang digunakan adalah fungsi *bruteForce* dan *DivAndConq*. Fungsi *bruteForce* melakukan iterasi untuk n titik sebanyak n(n-1)/2 perhitungan untuk jarak setiap titik yang ada dengan lainnya. Hasil perhitungan tersebut kemudian selalu dibandingkan apakah nilainya merupakan jarak terpendek. Apabila iya, akan digunakan sebagai perbandingan untuk pengujian titik - titik lainnya. Setiap perhitungan ini berdasarkan besar dimensi titik yang dipilih, semakin banyak dimensinya, maka ditambahkan lagi berdasarkan selisih jarak untuk setiap nilai pada setiap sumbu atau setiap dimensinya.

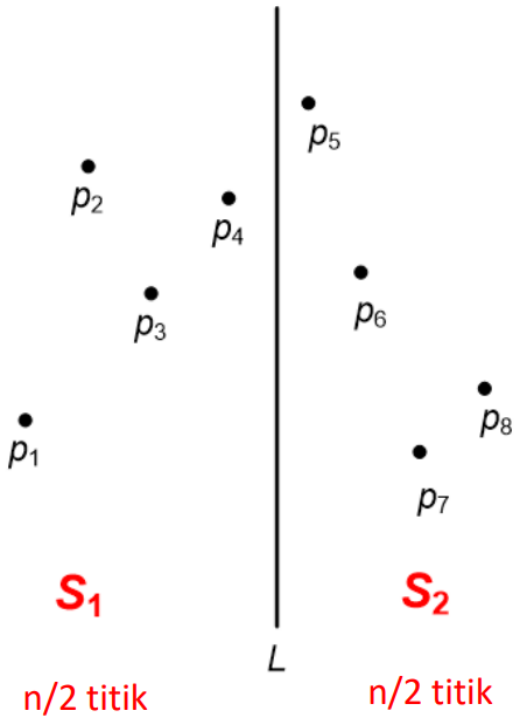
Untuk fungsi *DivAndConq*, fungsi ini merupakan implementasi dari algoritma *Divide and Conquer* itu sendiri. Pertama, akan dilakukan *Quicksort* pada list acak keseluruhan titik, kemudian pada algoritma ini, dilakukan pembagian list hingga dapat diselesaikan secara *brute force* dan kemudian dikomparasikan lagi masing - masing jarak yang didapatkan dan disimpan

jarak terkecil yang didapatkan. Setelah didapatkan jarang terkecil, dilakukan pengecekkan pada daerah *strip* yang membagi dua list tersebut, untuk setiap titik apabila jarak absis titik tersebut dengan titik tengah yang digunakan sebagai pembagi list menjadi dua lebih dekat dari jarak terdekat yang telah ditemukan, maka data tersebut akan ditampung terlebih dahulu ke dalam sebuah array. Hasil seluruh titik yang dimasukkan dalam sebuah array *strip* tersebut kemudian untuk sejumlah titik tersebut, dengan cara *bruteforce* dihitung kembali apabila ada jarak antar titik yang lebih dekat dari hasil pembagian, maka

Strategi

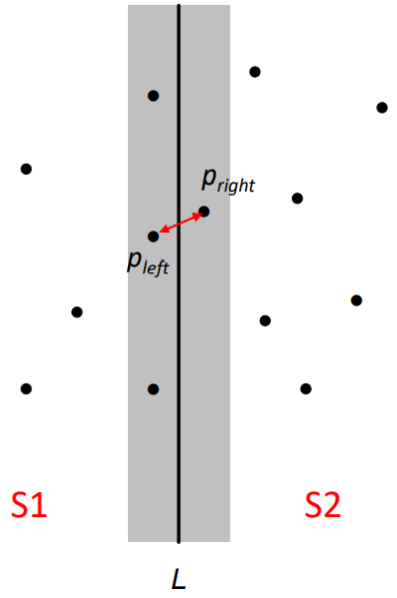
3.1 Strategi Algoritma Divide and Conquer

Strategi yang digunakan dalam mencari titik terdekat dari sekumpulan titik sembarang pada bidang tiga dimensi ini menggunakan pendekatan *divide and conquer* dengan cara membagi partisi titik berdasarkan salah satu sumbu pada bidang kartesius Sebelum pembagian ini, pertama kumpulan titik yang acak dalam *list* of titik tersebut diurutkan berdasarkan salah satu sumbu supaya bisa dipartisi menggunakan *quick sort* pada sumbu x terurut membesar, sehingga titik tengah dari *list* tersebut merupakan nilai titik tengah yang membagi dua list tersebut secara merata. Kemudian, secara rekursif, *list* tersebut dibagi dan dicari jarak terdekat dari setiap *list*. Pada setiap pembagian dibagi menjadi *list* kiri dan *list* kanan yang kemudian pasangan titik dengan jarak euclidean terdekatnya dihitung dan dibandingkan jarak euclidean dari kedua pasangan titik hasil dari *list* kiri dan *list* kanan untuk mendapatkan jarak terkecilnya.



Gambar 1 Hasil Pembagian Daerah setelah *Quicksort*

Setelah dihitung jarak terkecil dari *list* of titik, dilakukan pengecekkan jarak tersebut pada sumbu yang dipilih dalam partisi, yaitu nilai x (yang telah di *quick sort*) dari titik tengah dengan jarak euclidean terdekat. Pengecekkan jarak ini dilakukan pada daerah *strip,* dimana partisi dalam strip itu sendiri dilakukan *quick sort* lagi pada sumbu yang berbeda misalkan sumbu y (kecuali untuk dimensi 1), dan dikomparasikan ulang jarak titik - titik dalam daerah strip dan apabila ditemukan pasangan titik dengan jarak lebih dekat dari daerah diluar strip, yaitu *min distance* yang sudah ditemukan, maka jarak terdekat ini yang akan dipakai sebagai hasil jarak euclidean terdekatnya.



Gambar 2 Pengukuran Jarak Pasangan Titik dalam Daerah *Strip*

3.2 Analisis Kompleksitas

Pada algoritma divide and conquer ini, pertama, dilakukan *quicksort* untuk partisi titik - titik yang ada dalam bidang dan kemudian setiap pasangan titik dilakukan pengukuran euclidean dan dikomparasikan ke seluruh jumlah titik yang ada misalkan sebanyak n, maka pembagian partisi dari n titik tersebut dibagi menjadi n/2 titik pada dua bagian. Apabila titik - titik hanya berjumlah 2, maka dapat langsung dihitung jaraknya dan disimpulkan. Namun, untuk kasus berjumlah n atau banyak, diasumsikan jumlah titik $n = 2^k$, maka jumlah pembagian hingga bisa dilakukan *conquer* atau perhitungannya sendiri adalah $^2\log n$ dengan jumlah perhitungan totalnya menjadi n/2 (yaitu jarak pasangan - pasangan titik yang terbentuk dari hasil *divide*) dan pada setiap pembagian tersebut, dilakukan perhitungan titik - titik pada daerah *strip* apabila lebih dekat sebanyak cn dimana c merupakan jumlah pasangan yang mungkin lebih dekat. Kompleksitasnya:

$(n/2).^2\log n \rightarrow T(n) = 2T(n/2) + cn = O(n\ ^2\log n)$

Sedangkan, pada algoritma brute force, perhitungan untuk n titik, diperlukan sejumlah n(n-1)/2 operasi perhitungan untuk menguji setiap pasangan titik yang mungkin. Maka, kompleksitas untuk algoritma *brute force* adalah $O(n^2)$. Dari hasil tersebut, dapat disimpulkan bahwa secara

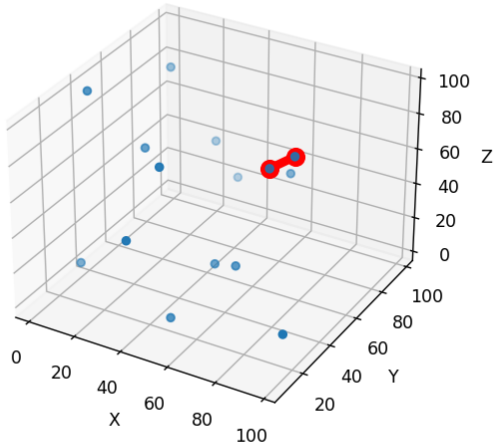
divide dan *conquer* algoritma memiliki kompleksitas yang lebih sedikit. Namun, kompleksitas pada *divide and conquer* tersebut didasarkan oleh pasangan titik berdimensi 2. Hal itu disebabkan oleh perhitungan jarak pada setiap pasangan titik. Sehingga, perhitungan pada setiap pembagian sejumlah pasangan $^{2}\log n$ dihitung sebanyak d-1 kali dimana d merupakan dimensi dari titik. Jumlah dimensi ini juga berdampak pada iterasi perhitungan jarak pada daerah *strip*. Sehingga, kompleksitas algoritma ini untuk setiap dimensi yang mungkin sebagai berikut.

$$(n/2).^{2}\log^{d-1} n \rightarrow T(n) = 2T(n/2, d-1) + U(n, d) = O(n\ (^{2}\log n)^{d-1}).$$

Berdasarkan perhitungan kompleksitas tersebut, algoritma ini juga memiliki keterbatasan untuk efisiensi pada dimensi yang cukup besar. Dalam implementasi yang diutamakan, yaitu pada dimensi 3, dimensi masih berdampak sangat amat kecil sehingga masih lebih efisien dibandingkan dengan algoritma brute force (perpangkatan 2). Namun, untuk dimensi yang besar, algoritma *bruteForce* memungkinkan untuk lebih efisien.

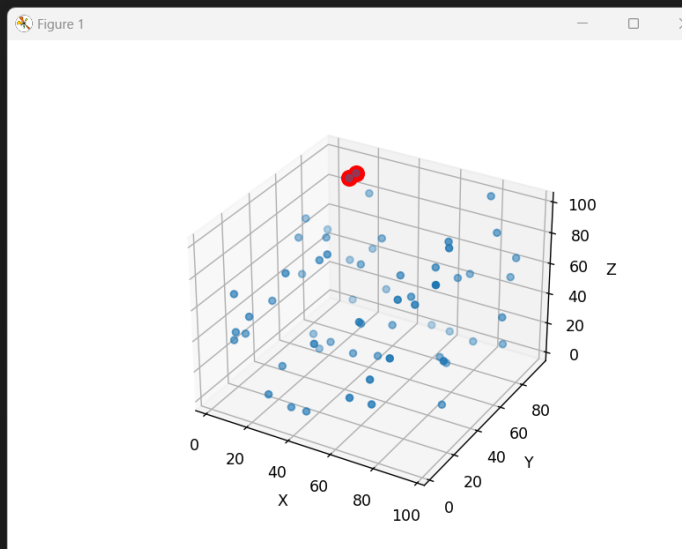
Testcase

```
Closest Pair of Points in 3D Using DIVIDE AND CONQUER Algorithm

Jumlah titik: 16
Dimensi: 3


Pasangan titik terdekat:  (Divide & Conquer)
Point 1: (70.47, 53.63, 71.08)
Point 2: (79.52, 56.02, 79.51)

Pasangan titik terdekat:  (bruteForce)
Point 1: (79.52, 56.02, 79.51)
Point 2: (70.47, 53.63, 71.08)

Jarak terdekat dengan Divide & Conquer:  12.596805150513367
Jarak terdekat dengan Brute Force:  12.596805150513367

Jumlah operasi euclidean dengan algoritma Divide & Conquer:  76
Jumlah operasi euclidean dengan algoritma Brute Force:  125

Waktu Eksekusi Divide & Conquer  0.0 miliseconds
Waktu Eksekusi Brute Force  0.0  miliseconds

Apakah ingin ditampilkan visualisasinya? (Y / N)Y
```



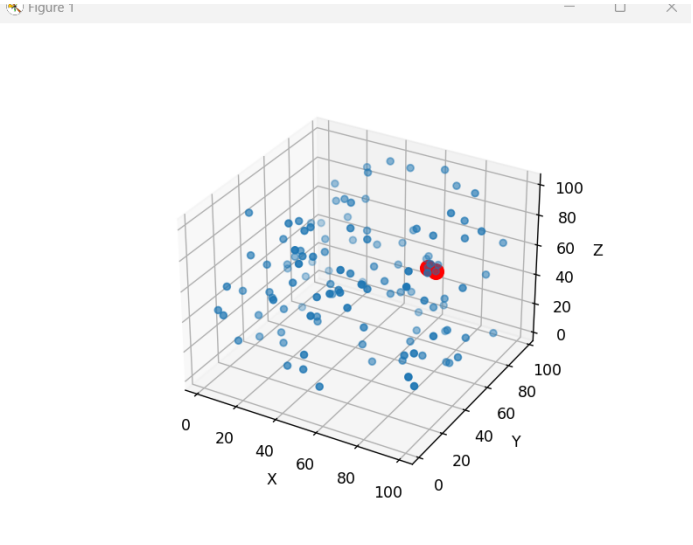
Gambar 3 Uji Kasus 1 Jumlah titik 16

```
Closest Pair of Points in 3D Using DIVIDE AND CONQUER Algorithm

Jumlah titik: 64
Dimensi: 3


Pasangan titik terdekat:  (Divide & Conquer)
Point 1: (14.07, 87.42, 93.07)
Point 2: (14.76, 92.01, 93.74)

Pasangan titik terdekat:  (bruteForce)
Point 1: (14.76, 92.01, 93.74)
Point 2: (14.07, 87.42, 93.07)

Jarak terdekat dengan Divide & Conquer:  4.689680159669743
Jarak terdekat dengan Brute Force:  4.689680159669743

Jumlah operasi euclidean dengan algoritma Divide & Conquer:  386
Jumlah operasi euclidean dengan algoritma Brute Force:  2026

Waktu Eksekusi Divide & Conquer  1.508951187133789 miliseconds
Waktu Eksekusi Brute Force  3.9825439453125  miliseconds

Apakah ingin ditampilkan visualisasinya? (Y / N)Y
```



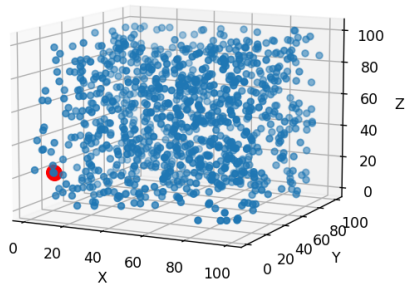
Gambar 4 Uji Kasus 2 Jumlah titik 64

```
Closest Pair of Points in 3D Using DIVIDE AND CONQUER Algorithm

Jumlah titik: 128
Dimensi: 3


Pasangan titik terdekat:  (Divide & Conquer)
Point 1: (60.91, 95.15, 30.8)
Point 2: (56.48, 95.52, 31.48)

Pasangan titik terdekat:  (bruteForce)
Point 1: (56.48, 95.52, 31.48)
Point 2: (60.91, 95.15, 30.8)

Jarak terdekat dengan Divide & Conquer:  4.497132419664779
Jarak terdekat dengan Brute Force:  4.497132419664779

Jumlah operasi euclidean dengan algoritma Divide & Conquer:  854
Jumlah operasi euclidean dengan algoritma Brute Force:  8144

Waktu Eksekusi Divide & Conquer  2.0051002502441406 miliseconds
Waktu Eksekusi Brute Force  12.006998062133789  miliseconds

Apakah ingin ditampilkan visualisasinya? (Y / N)Y
```



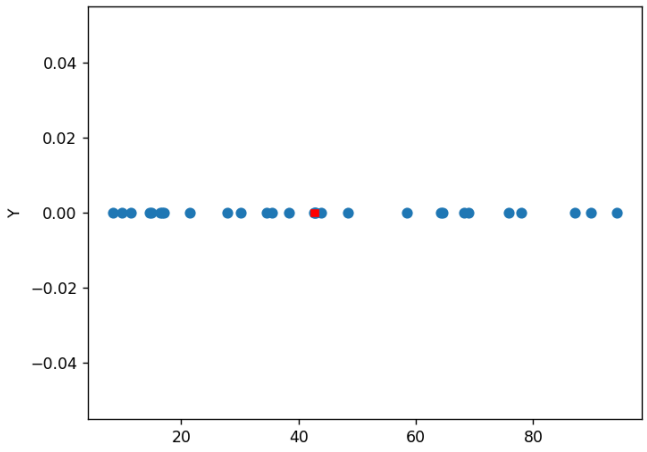
Gambar 5 Uji Kasus 3 Jumlah titik 128

```
Closest Pair of Points in 3D Using DIVIDE AND CONQUER Algorithm

Jumlah titik: 1000
Dimensi: 3


Pasangan titik terdekat:  (Divide & Conquer)
Point 1: (7.23, 8.84, 21.26)
Point 2: (6.47, 9.32, 21.48)

Pasangan titik terdekat:  (bruteForce)
Point 1: (6.47, 9.32, 21.48)
Point 2: (7.23, 8.84, 21.26)

Jarak terdekat dengan Divide & Conquer:  0.9254188241007425
Jarak terdekat dengan Brute Force:  0.9254188241007425

Jumlah operasi euclidean dengan algoritma Divide & Conquer:  7485
Jumlah operasi euclidean dengan algoritma Brute Force:  499512

Waktu Eksekusi Divide & Conquer  20.20096778869629 miliseconds
Waktu Eksekusi Brute Force  658.480167388916  miliseconds

Apakah ingin ditampilkan visualisasinya? (Y / N)Y
```

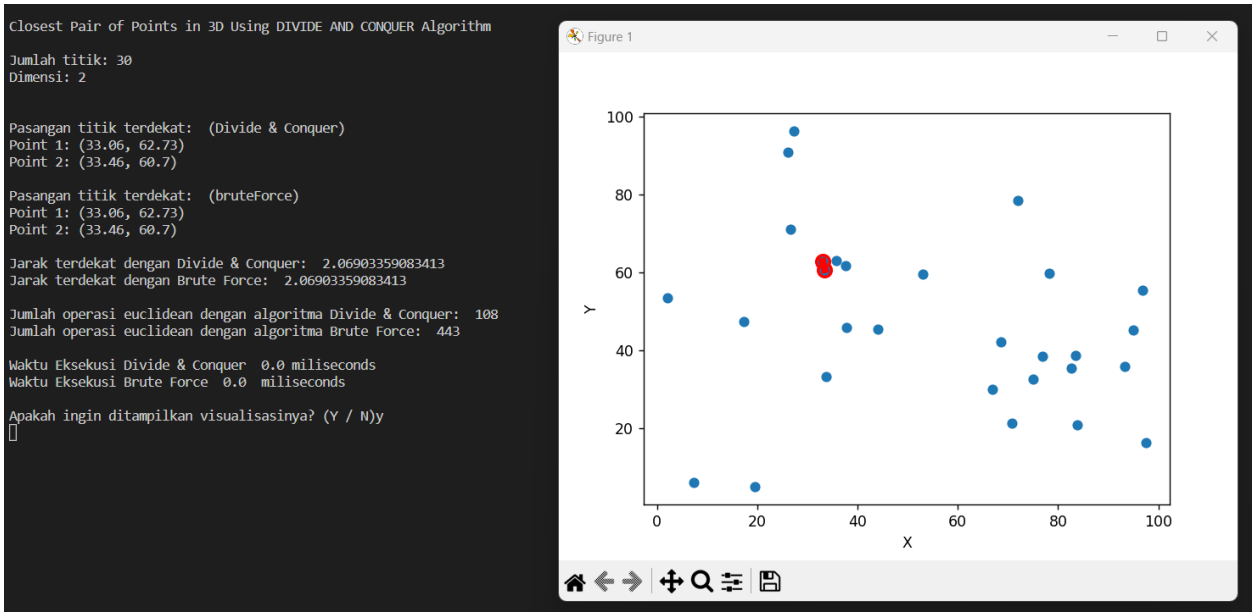


Gambar 6 Uji Kasus 4 Jumlah titik 1000

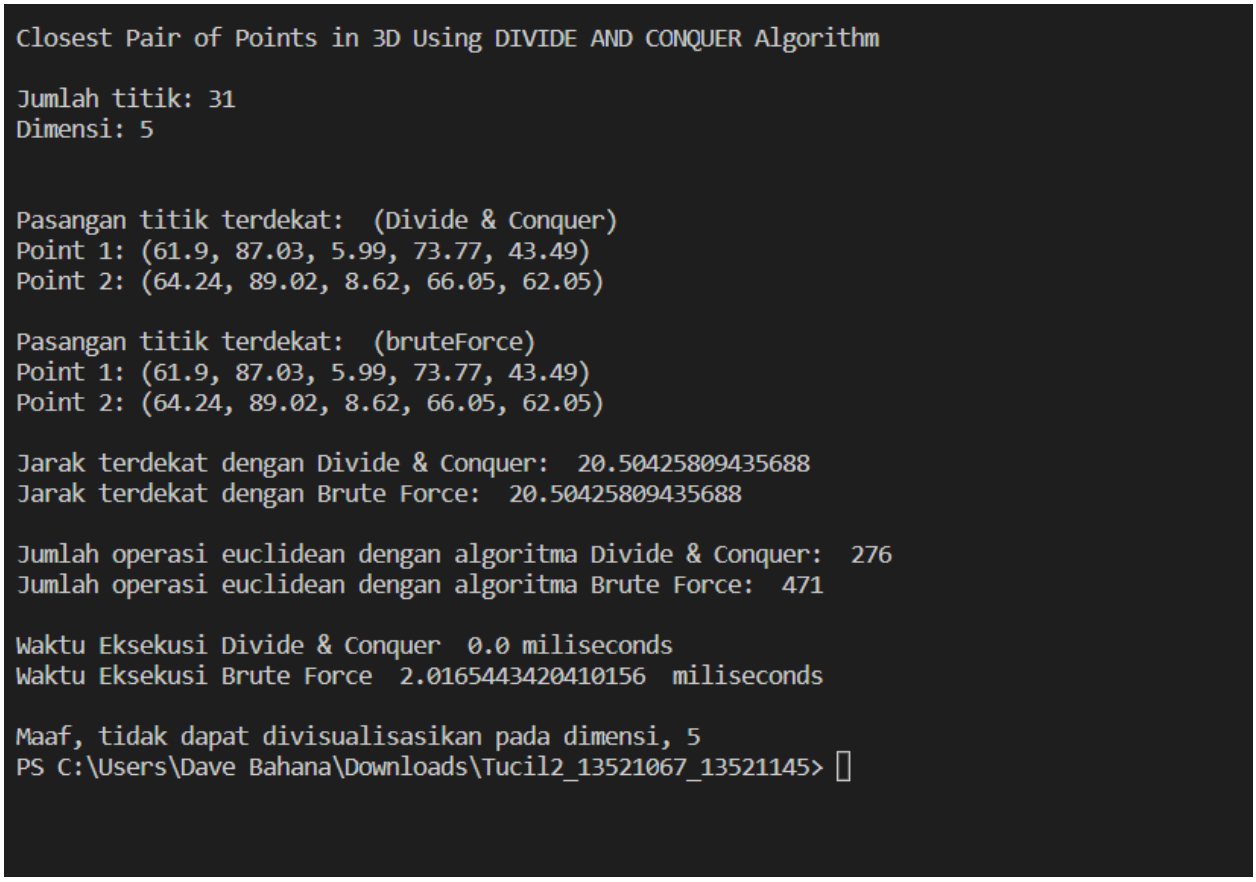
```
Closest Pair of Points in 3D Using DIVIDE AND CONQUER Algorithm

Jumlah titik: 30
Dimensi: 1


Pasangan titik terdekat:  (Divide & Conquer)
Point 1: (42.65)
Point 2: (42.72)

Pasangan titik terdekat:  (bruteForce)
Point 1: (42.72)
Point 2: (42.65)

Jarak terdekat dengan Divide & Conquer:  0.07000000000000028
Jarak terdekat dengan Brute Force:  0.07000000000000028

Jumlah operasi euclidean dengan algoritma Divide & Conquer:  91
Jumlah operasi euclidean dengan algoritma Brute Force:  441

Waktu Eksekusi Divide & Conquer  0.0 miliseconds
Waktu Eksekusi Brute Force  0.0  miliseconds

Apakah ingin ditampilkan visualisasinya? (Y / N)Y
```



Gambar 7 Uji Kasus 5 Jumlah titik 30 dimensi 1



Gambar 8 Uji Kasus 6 Jumlah titik 30 dimensi 2



Gambar 9 Uji Kasus 7 Jumlah titik 31 dimensi 5

```
Closest Pair of Points in 3D Using DIVIDE AND CONQUER Algorithm

Jumlah titik: 37
Dimensi: 6


Pasangan titik terdekat:  (Divide & Conquer)
Point 1: (34.16, 85.2, 43.4, 93.43, 64.31, 35.76)
Point 2: (18.44, 85.67, 28.31, 82.02, 56.29, 35.88)

Pasangan titik terdekat:  (bruteForce)
Point 1: (18.44, 85.67, 28.31, 82.02, 56.29, 35.88)
Point 2: (34.16, 85.2, 43.4, 93.43, 64.31, 35.76)

Jarak terdekat dengan Divide & Conquer:  25.876056500170193
Jarak terdekat dengan Brute Force:  25.876056500170193

Jumlah operasi euclidean dengan algoritma Divide & Conquer:  578
Jumlah operasi euclidean dengan algoritma Brute Force:  673

Waktu Eksekusi Divide & Conquer  1.5153884887695312 miliseconds
Waktu Eksekusi Brute Force  1.9948482513427734  miliseconds

Maaf, tidak dapat divisualisasikan pada dimensi, 6
PS C:\Users\Dave Bahana\Downloads\Tucil2_13521067_13521145>
```

Gambar 10 Uji Kasus 8 Jumlah titik 37 dimensi 6

KESIMPULAN, SARAN, DAN REFLEKSI

Dari Tugas Kecil II IF2211 Strategi Algoritma Semester II 2022/2023 dengan judul *Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer,* kami mendapati bahwa untuk mencari pasangan titik terdekat baik itu 2D maupun 3D tidak hanya dapat dilakukan dengan menggunakan pendekatan *brute force,* melainkan juga dengan pendekatan *divide and conquer*. Algoritma *Divide and Conque*r merupakan salah satu metode yang efektif dan efisien dalam mencari jarak terdekat antara sekumpulan titik pada bidang tiga dimensi. Dalam implementasinya, digunakan beberapa fungsi seperti *quicksort* dan *brute force* yang mempercepat perhitungan dan pengujian titik-titik pada list of titik. Dengan kompleksitas waktu O(n log n), algoritma ini cukup efisien dalam menyelesaikan permasalahan jarak terdekat.

Saran-saran yang dapat kami berikan untuk Tugas Kecil II IF2211 Strategi Algoritma Semester II 2022/2023 adalah terdapat salah satu cara untuk meningkatkan performa dari algoritma *divide and conquer* yaitu dengan melakukan paralelisasi. Dalam hal ini, algoritma *divide and conquer* dapat dibagi menjadi beberapa bagian yang dapat dijalankan secara paralel pada beberapa core atau mesin yang berbeda. Hal ini akan mempercepat waktu eksekusi dan mengurangi waktu yang dibutuhkan untuk menyelesaikan perhitungan. Selain itu, sebagai bagian dari penelitian lebih lanjut, disarankan untuk melakukan perbandingan antara algoritma *divide and conquer* dengan algoritma lainnya disamping algoritma *brute force* pada kumpulan titik-titik dalam bidang tiga dimensi. Hal ini dapat memberikan gambaran yang lebih jelas mengenai kelebihan dan kekurangan dari masing-masing algoritma. Kemudian, perlu dilakukan optimasi pada fungsi *quicksort* agar dapat bekerja secara efisien karena fungsi tersebut adalah salah satu fungsi yang krusial dalam algoritma *divide and conquer*. Salah satu cara untuk melakukan optimasi adalah dengan memilih pivot secara acak sehingga dapat mengurangi kemungkinan terjadinya kasus terburuk pada algoritma *quicksort*. Terakhir, program ini dapat dipublikasikan setelah dikembangkan lebih lanjut agar dapat bermanfaat menjadi referensi publik.

Setelah menyelesaikan Tugas Kecil II IF2211 Strategi Algoritma Semester II 2022/2023, kami dapat merefleksikan bahwa komunikasi antaranggota kelompok berjalan cukup baik sehingga tidak terjadi miskomunikasi dan kesalahpahaman dalam pengerjaan tugas kecil ini. Sebelum dimulainya pengerjaan tugas besar ini juga kami melakukan diskusi untuk membahas pembagian kerja untuk setiap anggota kelompok. Oleh karena itu, kami mendapatkan hasil yang sesuai dengan tujuan dan sasaran yang telah ditetapkan pada awal pengerjaan.

Lampiran

Repository Github: https://github.com/kenndave/Tucil2_13521067_13521145.git

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