

# **PENCARIAN TITIK TERDEKAT MENGGUNAKAN ALGORITMA DIVIDE AND CONQUER**

## **LAPORAN TUGAS KECIL**

**Disusun untuk memenuhi salah satu tugas kecil**

**mata kuliah Strategi Algoritma**

**IF2211-03**

**Oleh**

**Raditya Naufal A.**

**13521022**



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

## DAFTAR ISI

DAFTAR ISI	1
BAB I	2
DESKRIPSI MASALAH	2
BAB II	3
DASAR TEORI	3
BAB III	4
BAB IV	5
4.1. Fungsi Closest_pair	5
4.2. Fungsi _closest_pair	5
4.3. Fungsi _closest_pair_brute_force	6
4.5. Fungsi closest_pair_brute_force	6
BAB V	8
EKSPERIMEN	8
N = 16	8
N = 64	8
N = 128	9
N = 1000	9
BAB VI	10
PENUTUP	10
6.1. Kesimpulan	10
LAMPIRAN	11

## **BAB I**

### **DESKRIPSI MASALAH**

Mencari sepasang titik terdekat pada bidang 3D dengan menggunakan algoritma divide and conquer. Masukan program adalah jumlah titik ( $n$ ) yang diinginkan dan koordinat titik-titik yang dibangkitkan secara acak. Setiap titik diwakili oleh tiga koordinat ( $x, y, z$ ). Tujuan dari program adalah untuk menemukan sepasang titik yang memiliki jarak terdekat satu sama lain. Jarak dua titik dihitung menggunakan rumus Euclidean dan dinyatakan dalam bentuk jarak euclidean.

## **BAB II**

### **DASAR TEORI**

Divide and conquer adalah suatu teknik dalam pemrograman dan algoritma untuk memecahkan masalah yang kompleks menjadi beberapa submasalah yang lebih sederhana, kemudian menyelesaikan masing-masing submasalah secara terpisah dan menggabungkan solusi-solusi submasalah tersebut untuk mendapatkan solusi akhir untuk masalah asal yang lebih kompleks.

Teknik divide and conquer dapat diterapkan pada berbagai jenis masalah seperti pencarian, pengurutan, perkalian matriks, pencocokan pola, dan banyak lagi.

Langkah-langkah umum dalam algoritma divide and conquer adalah sebagai berikut:

1. Divide: Pecah masalah menjadi submasalah yang lebih kecil dan lebih sederhana.
2. Conquer: Selesaikan submasalah secara terpisah, baik dengan menggunakan algoritma rekursif atau algoritma iteratif.
3. Combine: Gabungkan solusi-solusi submasalah untuk memperoleh solusi akhir untuk masalah asal yang lebih kompleks.

Dalam praktiknya, teknik divide and conquer sering digunakan untuk mengurangi kompleksitas waktu dalam menyelesaikan masalah, karena mengurangi ukuran masalah dengan membaginya menjadi submasalah yang lebih kecil dapat mempercepat waktu eksekusi program.

Namun, perlu diingat bahwa tidak semua masalah dapat diselesaikan dengan teknik divide and conquer, dan pada beberapa kasus, teknik ini dapat menghasilkan kompleksitas yang sama atau bahkan lebih buruk dibandingkan dengan algoritma yang lebih sederhana.

### **BAB III**

#### **ANALISIS PERSOALAN**

Algoritma yang digunakan dalam kode ini adalah algoritma Divide and Conquer yang digunakan untuk mencari pasangan titik terdekat pada bidang dua dimensi. Algoritma ini bekerja dengan membagi himpunan titik menjadi dua bagian yang lebih kecil dan kemudian mencari pasangan titik terdekat di masing-masing bagian tersebut secara rekursif. Setelah itu, pasangan titik terdekat dari kedua bagian akan dicari untuk menemukan pasangan titik terdekat secara keseluruhan.

Pertama-tama, algoritma mengurutkan himpunan titik berdasarkan koordinat x. Kemudian, jika jumlah titik dalam himpunan tersebut kurang dari atau sama dengan 3, algoritma akan menggunakan metode brute force untuk mencari pasangan titik terdekat. Jika jumlah titik lebih besar dari 3, algoritma akan membagi himpunan titik menjadi dua bagian dan mencari pasangan titik terdekat di setiap bagian secara rekursif.

Setelah mendapatkan pasangan titik terdekat dari masing-masing bagian, algoritma akan mencari pasangan titik terdekat yang memiliki satu titik di setiap bagian. Untuk melakukan ini, algoritma akan menentukan garis tengah dan mengambil himpunan titik yang terletak di sekitar garis tersebut. Algoritma kemudian memeriksa setiap pasangan titik dalam himpunan ini dan mencari pasangan titik terdekat secara keseluruhan.

Untuk memeriksa pasangan titik terdekat, algoritma menggunakan metode brute force jika jumlah titik dalam himpunan tersebut kurang dari atau sama dengan 3. Jika jumlah titik lebih besar dari 3, algoritma akan menggunakan metode yang sama seperti sebelumnya, yaitu mencari pasangan titik terdekat di setiap himpunan secara brute force.

Selama proses pencarian pasangan titik terdekat, algoritma juga menghitung jumlah operasi yang dilakukan dengan menghitung jumlah iterasi yang dilakukan dalam loop. Jumlah operasi ini kemudian dapat digunakan untuk mengevaluasi kecepatan algoritma.

## BAB IV

### IMPLEMENTASI FUNGSI

#### 4.1. Fungsi `Closest_pair`

Fungsi ini berfungsi untuk melakukan sorting pada array points yang kemudian akan dioperasikan dengan algoritma *divide and conquer*

```
def closest_pair(points):  
    points_sorted_x = quick_sort(points)  
    return _closest_pair(points_sorted_x)
```

#### 4.2. Fungsi `_closest_pair`

Fungsi ini digunakan untuk mencari jarak titik terdekat dengan menggunakan metode *divide and conquer*.

```
def _closest_pair(points_sorted_x):  
    global c  
    d=len(points_sorted_x[0])  
    n = len(points_sorted_x)  
    if n <= 3:  
        return _closest_pair_brute_force(points_sorted_x)  
  
    # Divide the points into two sets  
    mid = n // 2  
    left_x, right_x = points_sorted_x[:mid], points_sorted_x[mid:]  
  
    # Recursively find the closest pair in the left and right sets  
    distance_left, closest_pair_left = _closest_pair(left_x)  
    distance_right, closest_pair_right = _closest_pair(right_x)  
    if distance_left < distance_right:  
        distance = distance_left  
        closest_pair = closest_pair_left  
    else:  
        distance = distance_right  
        closest_pair = closest_pair_right  
  
    # Find the closest pair with one point in each set  
  
    x_mid_point = points_sorted_x[mid]  
    strip = [point for point in points_sorted_x if abs(point[0] - x_mid_point[0]) <= distance]  
    n_strip = len(strip)
```

```

for i in range(n_strip):
    j = i + 1
    while j < n_strip and (strip[j][0] - strip[i][0]) < distance:
        distance_ij = np.sqrt(np.sum([(strip[i][k] - strip[j][k])**2 for k in
range(d)]))
        c+=1
        if distance_ij < distance:
            distance = distance_ij
            closest_pair = strip[i], strip[j]
        j += 1
return distance, closest_pair

```

#### 4.3. Fungsi `_closest_pair_brute_force`

Fungsi ini digunakan untuk mencari 2 point terdekat pada kondisi point  $\leq 3$

```

def _closest_pair_brute_force(points):
    global c
    closest_distance = float('inf')
    closest_points = None
    for i, point1 in enumerate(points):
        for point2 in points[i+1:]:
            distance = np.sum([(point1[i] - point2[i]) ** 2 for i in
range(len(point1))]) ** 0.5
            c+=1
            if distance < closest_distance:
                closest_distance = distance
                closest_points = (point1, point2)
    return closest_distance, closest_points

```

#### 4.5. Fungsi `closest_pair_brute_force`

Fungsi ini digunakan untuk mencari 2 point terdekat dari array points dengan menggunakan algoritma brute force.

```

def closest_pair_brute_force(points):
    global cb
    closest_distance = float('inf')
    closest_points = None
    for i, point1 in enumerate(points):
        for point2 in points[i+1:]:
            distance = np.sum([(point1[i] - point2[i]) ** 2 for i in
range(len(point1))]) ** 0.5

```

```

        cb+=1
        if distance < closest_distance:
            closest_distance = distance
            closest_points = (point1, point2)
    return closest_distance, closest_points

```

#### 4.6 Fungsi quick\_sort

Fungsi ini menggunakan algoritma quick sort untuk mengurutkan array berdasarkan member pertama dalam tiap subarray

```

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2][0]
    left = []
    right = []
    equal = []
    for elem in arr:
        if elem[0] < pivot:
            left.append(elem)
        elif elem[0] == pivot:
            equal.append(elem)
        else:
            right.append(elem)
    return quick_sort(left) + equal + quick_sort(right)

```



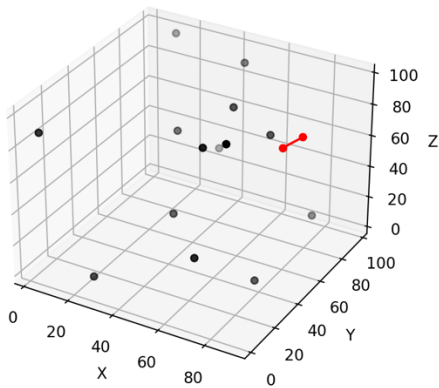
## BAB V

### EKSPERIMEN

Semua eksperimen dilakukan pada Apple silicone m1

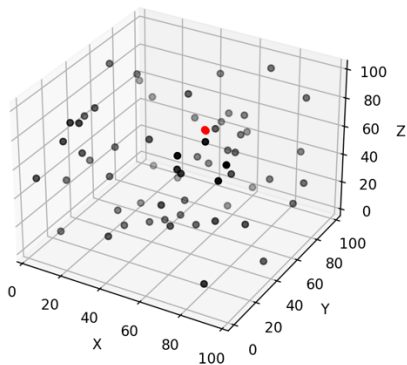
**N = 16**

```
Numbers of element(2-1000) : 16  
Numbers of dimension (3-7) : 3  
The closest pair of points is [85.5, 47.11, 84.43] and [90.49, 54.05, 88.59] with a distance of 9.51  
Time: 1.23 ms  
Total euclidian equation: 61
```



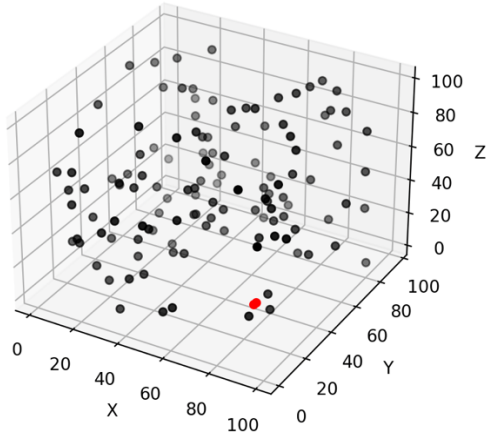
**N = 64**

```
Numbers of element(2-1000) : 64  
Numbers of dimension (3-7) : 3  
The closest pair of points is [47.38, 76.6, 53.29] and [47.62, 75.25, 55.07] with a distance of 2.25  
Time: 7.479999999999995 ms  
Total euclidian equation: 532
```



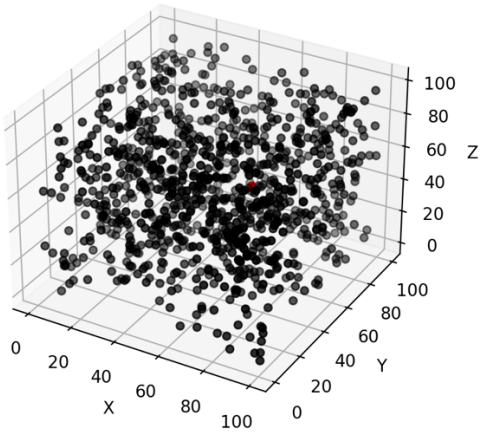
**N = 128**

```
Numbers of element(2-1000) : 128
Numbers of dimension (3-7) : 3
The closest pair of points is [92.73, 4.87, 31.7] and [93.71, 4.57, 33.72] with a distance of 2.27
Time: 19.36 ms
Total euclidian equation: 1935
```



**N = 1000**

```
Numbers of element(2-1000) : 1000
Numbers of dimension (3-7) : 3
The closest pair of points is [72.94, 41.81, 65.22] and [73.1, 42.08, 65.84] with a distance of 0.69
Time: 151.45 ms
Total euclidian equation: 37864
```



## **BAB VI**

### **PENUTUP**

#### **6.1. Kesimpulan**

Program untuk menentukan closest pair disini untuk menghitung jarak dalam ruang 3 dimensi sangat menghemat jumlah operasi yang dilakukan oleh algoritma bruteforce. Dalam algoritma divide and conquer jumlah operasi setiap program ini dijalankan akan berbeda karena perbedaan kordinat titik yang dihasilkan.

## LAMPIRAN

Link Repository Github : [https://github.com/Raditss/Tucil2\\_13521022](https://github.com/Raditss/Tucil2_13521022)

POIN	YA	TIDAK
Program berhasil dikompilasi tanpa kesalahan		
Program berhasil <i>running</i>		
Program dapat menerima masukan dan memberikan keluaran		
Keluaran program sudah benar (solusi closest pair benar)		
Bonus 1 dikerjakan		
Bonus 2 dikerjakan		