LAPORAN TUGAS KECIL 2 IF2211 STRATEGI ALGORITMA
SEMESTER II TAHUN 2022/2023

MENCARI PASANGAN TITIK TERDEKAT 3D dengan ALGORITMA DIVIDE AND CONQUER

Disusun oleh :

Jimly Firdaus 13521102

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2023

DAFTAR ISI

BAB I

ALGORITMA DIVIDE AND CONQUER SECARA GARIS BESAR

1.1 Pengertian Algoritma *Divide and Conquer*

Algoritma divide and conquer merupakan algoritma yang membagi persoalan menjadi beberapa upa-persoalan lebih kecil yang memiliki kemiripan dengan persoalan semula. Cara kerja algoritma ini adalah dengan menyelesaikan upa-persoalan lalu menggabungkan solusi dari upa-persoalan tersebut.

Contoh persoalan yang dapat diselesaikan dengan memanfaatkan algoritma divide and conquer :

1. Persoalan MinMaks
2. Menghitung perpangkatan
3. *Sorting (Mergesort & Quicksort)*
4. Mencari pasangan titik terdekat (*shortest pair*)
5. *Convex Hull*
6. Perkalian Matriks
7. Perkalian bilangan bulat besar
8. Perkalian dua buah polinom

1.2 *Shortest Pair* dan Kaitannya dengan Algoritma *Divide and Conquer*

Shortest Pair adalah istilah dalam pemrograman komputer yang mengacu pada pasangan titik terdekat dalam sebuah himpunan titik. Algoritma *Divide and Conquer* (D&C) merupakan metode pemecahan masalah yang umum digunakan dalam pemrograman komputer, dimana masalah besar dipecah menjadi sub-masalah yang lebih kecil dan kemudian sub-masalah tersebut diselesaikan secara rekursif hingga mencapai ukuran yang cukup kecil sehingga dapat dipecahkan dengan mudah.

Dalam konteks pencarian *Shortest Pair*, algoritma D&C dapat digunakan untuk menyelesaikan masalah dengan kompleksitas yang tinggi. Salah satu pendekatan yang umum digunakan dalam algoritma D&C untuk menemukan pasangan titik terdekat adalah dengan membagi himpunan titik menjadi dua bagian yang sama besar, kemudian

mencari pasangan titik terdekat di setiap bagian tersebut secara rekursif. Setelah itu, jarak terdekat di antara kedua pasangan titik terdekat dari kedua bagian tersebut dicari.

Metode ini memiliki keuntungan yang signifikan dalam hal kecepatan dan efisiensi karena setiap sub-masalah yang dipecahkan dapat dihitung secara paralel dan independen. Dengan cara ini, algoritma D&C dapat menyelesaikan masalah *Shortest Pair* dalam waktu yang relatif singkat bahkan untuk himpunan titik yang sangat besar. Oleh karena itu, algoritma D&C sering digunakan dalam aplikasi praktis seperti navigasi, pemetaan, pengolahan gambar, dan analisis data.

1.3 Deskripsi Masalah *Shortest Pair* pada Tugas Kecil 2 IF2211 Strategi Algoritma

Pada Tugas Kecil 2 IF2211 Strategi Algoritma, masalah yang diberikan adalah mengimplementasikan algoritma *divide and conquer* untuk permasalahan *shortest pair* pada bidang 3 dimensi dengan spesifikasi:

Masukan program:

1. n
2. Titik-titik (dibangkitkan secara acak) dalam koordinat (x,y,z)

Luaran program:

1. Sepasang titik yang jaraknya terdekat dan nilai jaraknya
2. Banyaknya operasi perhitungan rumus Euclidian
3. Waktu riil dalam detik (spesifikasikan komputer yang digunakan)

Spesifikasi komputer yang digunakan:

1. CPU : Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
2. RAM : 16.0 GB (15.8 GB usable)
3. System Type : 64-bit operating system, x64-based processor

BAB II

PENJELASAN ALGORITMA DIVIDE AND CONQUER YANG DIGUNAKAN

Algoritma *Divide and Conquer* yang digunakan adalah dengan memecah setiap *array of Points* dalam setiap pemanggilan *divideAndConquer* secara rekursif. Basis pada fungsi rekursif ini adalah untuk jumlah elemen pada *array* sebanyak 2 dan 3. Jika jumlah elemen sebanyak 2, maka fungsi me-*return* sebuah object *ClosestPair* yang berisi *euclidean distance* dari *pair* tersebut serta pasangan titik tersebut. Jika jumlah elemen sebanyak 3, maka akan dicek terlebih dahulu semua kombinasi dari ketiga titik itu untuk dicari *euclidean distance* terkecil sehingga terjadi 3 kali pemanggilan fungsi untuk mencari *euclidean distance* dan kemudian akan me-*return* hal yang sama (jarak terkecil serta *pair*-nya).

Untuk jumlah elemen > 3, akan dilakukan *sort* terlebih dahulu berdasarkan *axis*. *Default*-nya, *axis* dimulai dari 0 (berupa *axis-x* atau r1 dalam *vector). Quicksort* yang dibuat menerima kumpulan dari *point* (*array of point*) dan *axis* yang akan dijadikan pembanding *sorting.* Untuk setiap pemanggilan rekursif, jika jumlah elemen *array* masih > 3, maka *quicksort* juga akan dipanggil untuk melakukan *sorting* kembali namun pada *axis* yang berbeda (*axis* + 1) hingga *axis* sama dengan dimensi – 1. Jika *axis* sudah mencapai dimensi – 1, maka *axis* akan di-*reset* kembali menjadi *axis-x* atau r1 jika rekursif masih berlanjut.

Perbandingan setiap *pair* berdasarkan *euclidean distance*-nya. Karena *array* dibagi menjadi sub-persoalan n/2 dalam setiap rekursif, maka pada basis akan mengembalikan *euclidean distance* yang kemudian akan dibandingkan dan dicari yang paling minimum. Selain itu juga dilakukan perhitungan untuk daerah *stripe* dimana disaat pembagian / pemecahan *array* menjadi n/2 bagian, terdapat 2 buat titik / lebih yang terpisah pada bagian yang berbeda namun bisa saja memiliki *euclidean distance* yang lebih minimum, sehingga dilakukan pengecekan pada daerah *stripe* tersebut dan kemudian dibandingkan dengan hasil *divide and conquer* sebelumnya.

BAB III

KODE PROGRAM DALAM BAHASA JAVA

2.1 Point.java

Berisi *class* Point sebagai *base type* untuk *vector* dimensi – n

```java
package type;

public class Point {
    protected double[] vector;
    protected int dimension;

    public Point (int d) {
        this.dimension = d;
        this.vector = new double[this.dimension];
    }

    public double[] getVector () {
        return this.vector;
    }

    public int getDimension () {
        return this.dimension;
    }

    public void printVector () {
        System.out.print(s: "(");
        for (int i = 0; i < this.vector.length; i++) {
            System.out.print(this.vector[i]);
            if (i == this.vector.length - 1) {
                System.out.print(s: "");
            } else {
                System.out.print(s: ", ");
            }
        }
        System.out.print(s: ")\n");
    }
}
```

2.2 ClosestPair.java

Berisi *class ClosestPair* sebagai *type* dari hasil komputasi (berisi *euclidean distance* dan *closest pair*)

```java
package type;

public class ClosestPair {
    private Point[] pair;
    private double distance;

    public void setPair (Point[] pair) {
        this.pair = pair;
    }

    public void setDistance (double d) {
        this.distance = d;
    }

    public Point[] getPair () {
        return this.pair;
    }

    public double getDistance () {
        return this.distance;
    }
}
```

2.3 QuickSort.java

Berisi *class QuickSort* yang memiliki *method* sorting dengan metode *quicksort. Quicksort* yang digunakan menerima array of *Point* dan melakukan sorting berdasarkan *axis* yang ditetapkan.

```java
import type.*;

public class QuickSort {
    public static void quicksort(Point[] points, int axis) {
        if (points == null || points.length == 0) {
            return;
        }
        sort(points, low: 0, points.length - 1, axis);
    }

    private static void sort(Point[] points, int low, int high, int axis) {
        if (low >= high) {
            return;
        }
        int partitionIndex = partition(points, low, high, axis);
        sort(points, low, partitionIndex - 1, axis);
        sort(points, partitionIndex + 1, high, axis);
    }

    private static int partition(Point[] points, int low, int high, int axis) {
        double pivot = points[high].getVector()[axis];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (points[j].getVector()[axis] <= pivot) {
                i++;
                swap(points, i, j);
            }
        }
        swap(points, i + 1, high);
        return i + 1;
    }

    private static void swap(Point[] points, int i, int j) {
        Point temp = points[i];
        points[i] = points[j];
        points[j] = temp;
    }
}
```

2.4 Points.java

Berisi *class Points* yang merupakan kumpulan dari *Point* dan memiliki *method divide and conquer* serta *brute force* untuk mencair *closest pair* dari kumpulan-kumpulan *point.*

```java
import java.util.Random;

import java.util.Arrays;
import type.*;

public class Points {
    private Point[] points;
    private int size;

    private final int min = -100;
    private final int max = 100;

    public static int divideAndConquer = 0;
    public static int bruteForce = 0;

    public static int getTotalDivideConquer () {
        return divideAndConquer;
    }

    public static int getTotalBruteForce () {
        return bruteForce;
    }

    Random random = new Random();

    public Points(int size, int dimension) {
        this.size = size;
        this.points = new Point[this.size];
        for (int i = 0; i < this.size; i++) {
            this.points[i] = new Point(dimension);
            for (int j = 0; j < dimension; j++) {
                this.points[i].getVector()[j] = random.nextDouble(max + 1) - min;
            }
        }
    }

    public Point[] getPoints() {
        return this.points;
    }

    public int getSize() {
        return this.size;
    }
```

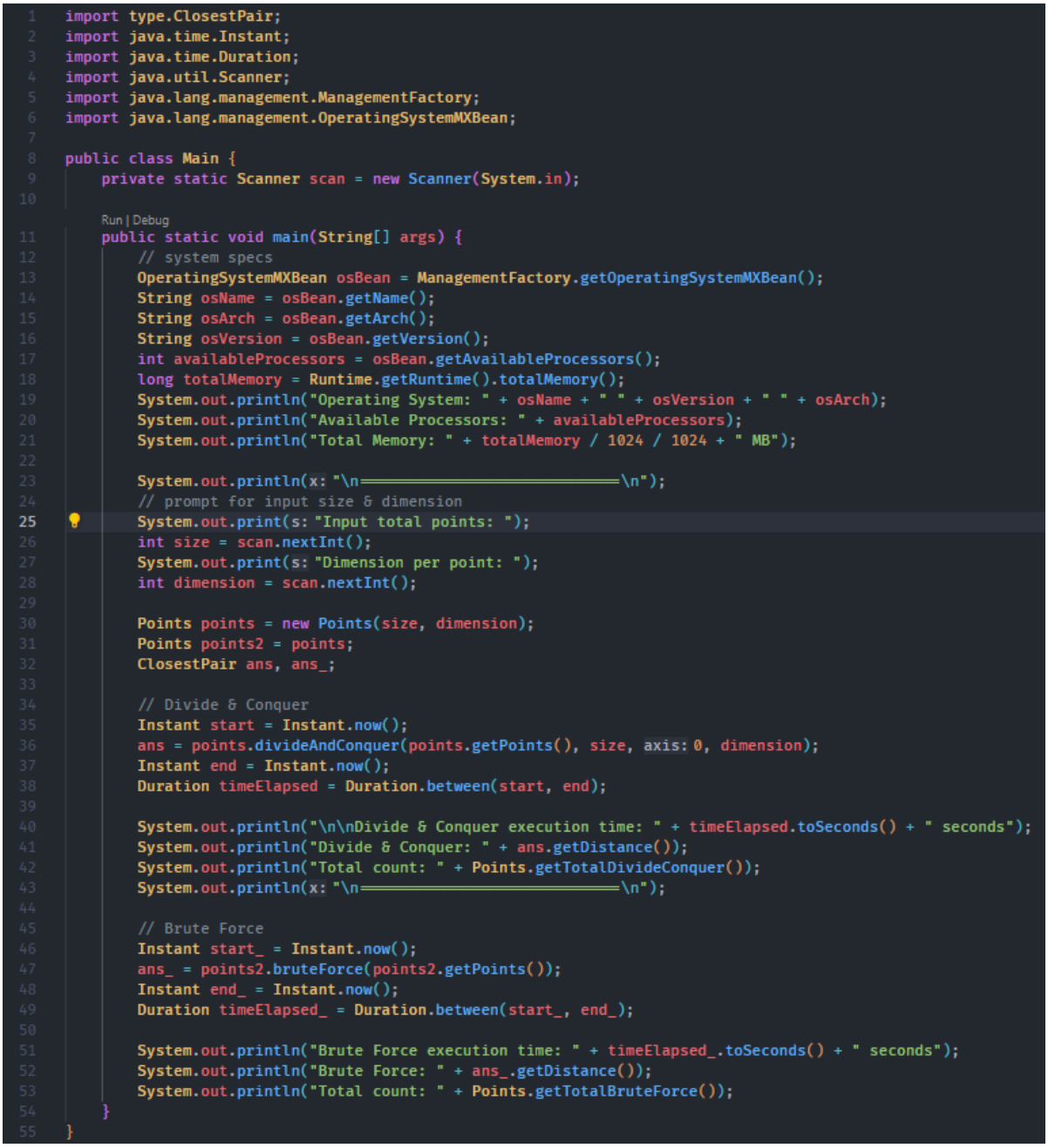
```java
    private static double euclidDistance (Point p1, Point p2, boolean div_conquer) {
        double distance = 0;
        for (int i = 0; i < p1.getDimension(); i++) {
            double diff = p1.getVector()[i] - p2.getVector()[i];
            distance += Math.pow(diff, b: 2);
        }
        if (div_conquer) {
            divideAndConquer++;
        } else {
            bruteForce++;
        }

        return Math.sqrt(distance);
    }

    public ClosestPair divideAndConquer(Point[] points, int arr_size, int axis, int max_dimension) {
        if (arr_size == 2) {
            ClosestPair res = new ClosestPair();
            res.setPair(points);
            res.setDistance(euclidDistance(points[0], points[1], div_conquer: true));
            divideAndConquer++;
            return res;
        }
        else if (arr_size == 3) {
            ClosestPair res = new ClosestPair();
            Point[] temp = new Point[2];
            double d1 = euclidDistance(points[0], points[1], div_conquer: true);
            double d2 = euclidDistance(points[1], points[2], div_conquer: true);
            double d3 = euclidDistance(points[0], points[2], div_conquer: true);
            divideAndConquer += 2;
            double minDist = Math.min(Math.min(d1, d2), d3);
            if (minDist == d1) {
                temp[0] = points[0];
                temp[1] = points[1];
                res.setDistance(d1);
            }
            else if (minDist == d2) {
                temp[0] = points[1];
                temp[1] = points[2];
                res.setDistance(d2);
            } else {
                temp[0] = points[0];
                temp[1] = points[2];
```

```
            temp[1] = points[2];
            res.setDistance(d3);
        }
        res.setPair(temp);

        return res;
    }
    else {
        axis = axis + 1 < max_dimension - 1 ? axis + 1 : 0;
        QuickSort.quicksort(points, axis);
        int start_idx = 0;
        int end_idx = arr_size % 2 == 0
            ? arr_size / 2
            : arr_size / 2 + 1;
        Point[] setOfPoints_1 = Arrays.copyOfRange(points, start_idx, end_idx);
        Point[] setOfPoints_2 = Arrays.copyOfRange(points, end_idx, arr_size);
        ClosestPair res_1 = divideAndConquer(setOfPoints_1, arr_size/2, axis, max_dimension);
        ClosestPair res_2 = divideAndConquer(setOfPoints_2, arr_size/2, axis, max_dimension);

        ClosestPair res = new ClosestPair();
        if (res_1.getDistance() > res_2.getDistance()) {
            res = res_2;
        } else {
            res = res_1;
        }
        double distanceInStrip = Double.MAX_VALUE;
        for (int i = 0; i < arr_size; i++) {
            for (int j = i + 1; j < arr_size; j++) {
                for (int k = 0; k < points[i].getDimension(); k++) {
                    if (Math.abs(points[i].getVector()[k] - points[j].getVector()[k]) > res.getDistance()) {

                    } else {
                        distanceInStrip = euclidDistance(points[i], points[j], div_conquer: true);
                        if (distanceInStrip < res.getDistance()) {
                            Point[] temp = new Point[2];
                            temp[0] = points[i];
                            temp[1] = points[j];
                            res.setPair(temp);
                            res.setDistance(distanceInStrip);
                        }
                    }
                }
            }
        }
```

```
                                    res.setDistance(distanceInStrip);
                                }
                            }
                        }
                    }
                }
                return res;
            }

        }

        public ClosestPair bruteForce (Point[] points) {
            double shortestDistance = Double.MAX_VALUE;
            ClosestPair res = new ClosestPair();
            Point[] temp = new Point[2];

            // Compare each pair of points
            for (int i = 0; i < points.length - 1; i++) {
                for (int j = i + 1; j < points.length; j++) {
                    double distance = euclidDistance(points[i], points[j], div_conquer: false);
                    bruteForce++;
                    if (distance < shortestDistance) {
                        shortestDistance = distance;
                        temp[0] = points[i];
                        temp[1] = points[j];
                    }
                }
            }
            res.setPair(temp);
            res.setDistance(shortestDistance);
            return res;
        }
    }

```

2.5 Main.java

Berisi *main program* sebagai *driver* dari tugas ini.

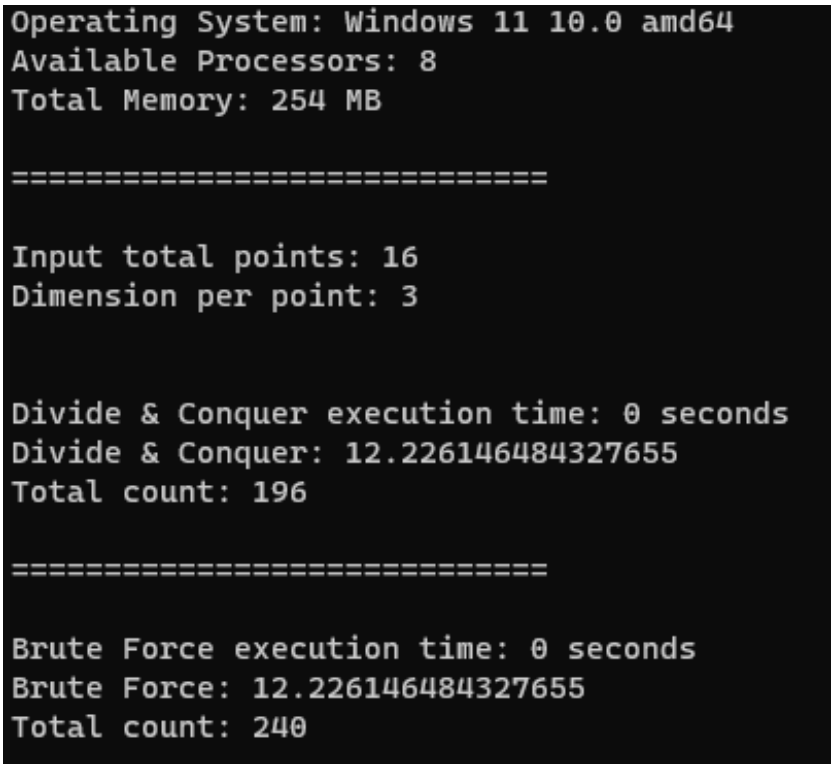


```java
import type.ClosestPair;
import java.time.Instant;
import java.time.Duration;
import java.util.Scanner;
import java.lang.management.ManagementFactory;
import java.lang.management.OperatingSystemMXBean;

public class Main {
    private static Scanner scan = new Scanner(System.in);

    Run | Debug
    public static void main(String[] args) {
        // system specs
        OperatingSystemMXBean osBean = ManagementFactory.getOperatingSystemMXBean();
        String osName = osBean.getName();
        String osArch = osBean.getArch();
        String osVersion = osBean.getVersion();
        int availableProcessors = osBean.getAvailableProcessors();
        long totalMemory = Runtime.getRuntime().totalMemory();
        System.out.println("Operating System: " + osName + " " + osVersion + " " + osArch);
        System.out.println("Available Processors: " + availableProcessors);
        System.out.println("Total Memory: " + totalMemory / 1024 / 1024 + " MB");

        System.out.println(x: "\n═══════════════════════════════\n");
        // prompt for input size & dimension
        System.out.print(s: "Input total points: ");
        int size = scan.nextInt();
        System.out.print(s: "Dimension per point: ");
        int dimension = scan.nextInt();

        Points points = new Points(size, dimension);
        Points points2 = points;
        ClosestPair ans, ans_;

        // Divide & Conquer
        Instant start = Instant.now();
        ans = points.divideAndConquer(points.getPoints(), size, axis: 0, dimension);
        Instant end = Instant.now();
        Duration timeElapsed = Duration.between(start, end);

        System.out.println("\n\nDivide & Conquer execution time: " + timeElapsed.toSeconds() + " seconds");
        System.out.println("Divide & Conquer: " + ans.getDistance());
        System.out.println("Total count: " + Points.getTotalDivideConquer());
        System.out.println(x: "\n═══════════════════════════════\n");

        // Brute Force
        Instant start_ = Instant.now();
        ans_ = points2.bruteForce(points2.getPoints());
        Instant end_ = Instant.now();
        Duration timeElapsed_ = Duration.between(start_, end_);

        System.out.println("Brute Force execution time: " + timeElapsed_.toSeconds() + " seconds");
        System.out.println("Brute Force: " + ans_.getDistance());
        System.out.println("Total count: " + Points.getTotalBruteForce());
    }
}
```

BAB IV

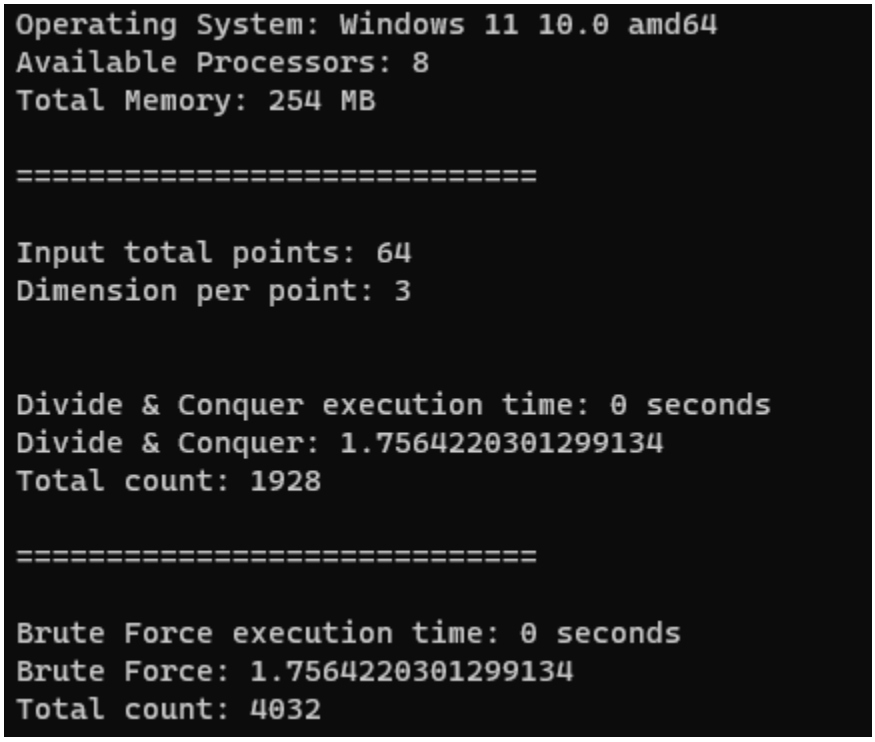
INPUT / OUTPUT PROGRAM

Test Case 1 (n = 16, dimensi = 3)



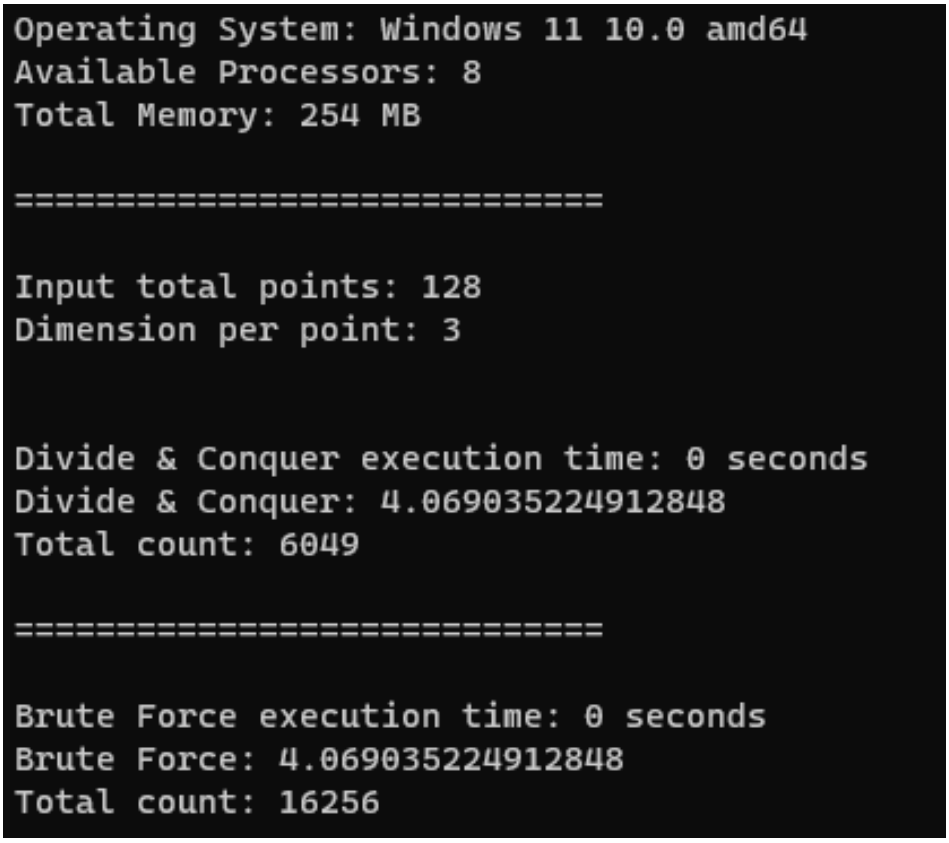
Untuk total titik berjumlah 16 dan ruang pada dimensi 3, Algoritma *divide and conquer* melakukan perhitungan *euclidean distance* sebanyak 196 kali dengan waktu eksekusi yang sama dengan algoritma *brute force* yang melakukan perhitungan *euclidean distance* sebanyak 240 kali.

Test Case 2 (n = 64, dimensi = 3)



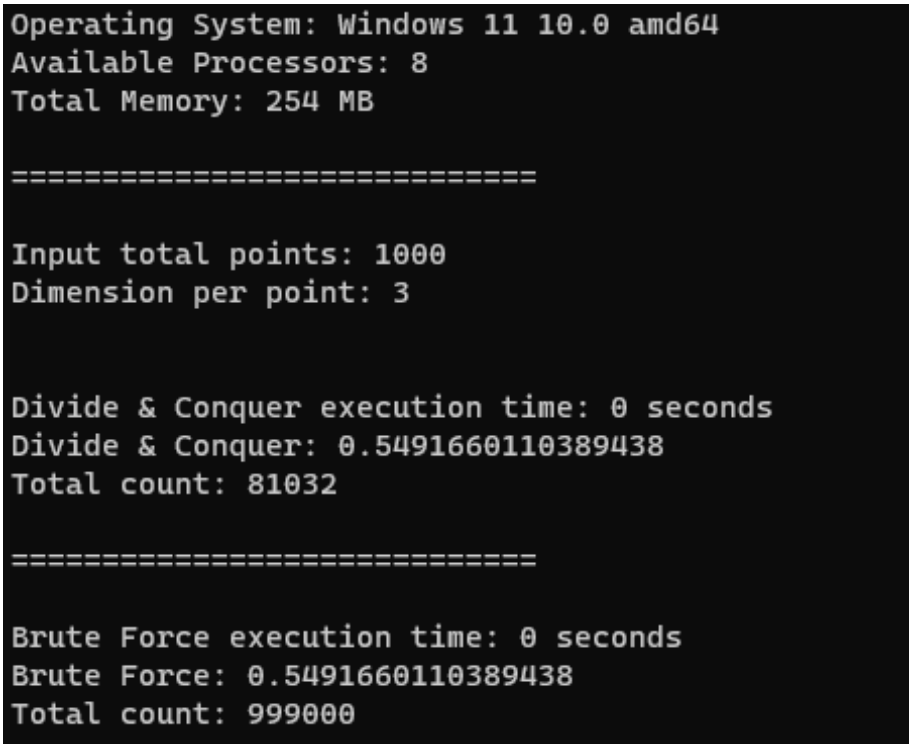
Untuk total titik berjumlah 64 dan ruang pada dimensi 3, Algoritma *divide and conquer* melakukan perhitungan *euclidean distance* sebanyak 1928 kali dengan waktu eksekusi yang sama dengan algoritma *brute force* yang melakukan perhitungan *euclidean distance* sebanyak 4032 kali. Hal ini berarti algoritma *divide and conquer* berhasil meminimasi perhitungan dengan *euclidean distance* sebanyak 52%

Test Case 3 (n = 128, dimensi = 3)



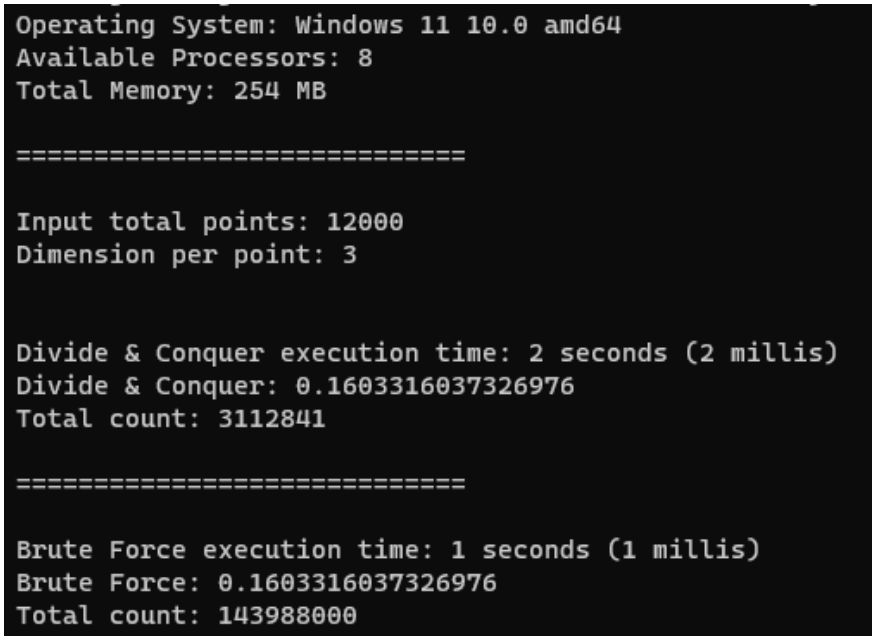
Untuk total titik berjumlah 128 dan ruang pada dimensi 3, Algoritma *divide and conquer* melakukan perhitungan *euclidean distance* sebanyak 6049 kali dengan waktu eksekusi yang sama dengan algoritma *brute force* yang melakukan perhitungan *euclidean distance* sebanyak 16256 kali. Hal ini berarti algoritma *divide and conquer* berhasil meminimasi perhitungan dengan *euclidean distance* sebanyak 62%

Test Case 4 (n = 1000, dimensi = 3)



Untuk total titik berjumlah 1000 dan ruang pada dimensi 3, Algoritma *divide and conquer* melakukan perhitungan *euclidean distance* sebanyak 81032 kali dengan waktu eksekusi yang sama dengan algoritma *brute force* yang melakukan perhitungan *euclidean distance* sebanyak 999000 kali. Hal ini berarti algoritma *divide and conquer* berhasil meminimasi perhitungan dengan *euclidean distance* sebanyak 91%

Test Case 5 (n = 135, dimensi 102)



Untuk total titik berjumlah 12000 dan ruang pada dimensi 3, Algoritma *divide and conquer* melakukan perhitungan *euclidean distance* sebanyak 3112841 kali dengan waktu eksekusi yang sama dengan algoritma *brute force* yang melakukan perhitungan *euclidean distance* sebanyak 143988000 kali. Hal ini berarti algoritma *divide and conquer* berhasil meminimasi perhitungan dengan *euclidean distance* sebanyak 98%

Hal ini menunjukkan bahwa perhitungan jarak pada algoritma *divide and conquer* lebih sedikit jika dibandingkan dengan algoritma *brute force*. Akan terlihat perbedaan yang sangat signifikan untuk n yang besar seperti n >= 1000. Waktu eksekusi algoritma *divide and conquer* dan *brute force* bergantung pada *case-case* tertentu misalnya untuk titik dengan dimensi k dan sejumlah n dimana k > n, maka algoritma *divide and conquer* akan memakan waktu yang sedikit lebih lama dibandingkan dengan *brute force.* Hal ini tidak terlepas dari spesifikasi perangkat yang digunakan.

BAB V

LINK KODE PROGRAM

Link *repository* : https://github.com/Jimly-Firdaus/Tucil2_13521102

BAB VI

TABEL PENILAIAN

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	√	
2. Program berhasil *running*	√	
3. Program dapat menerima masukan dan menuliskan keluaran	√	
4. Luaran program sudah benar (solusi *closest pair* benar)	√	
5. Bonus 1 dikerjakan		√ (ribet visualisasi dengan Java)
6. Bonus 2 dikerjakan	√	