

**LAPORAN TUGAS KECIL 2**  
**IF2211 STRATEGI ALGORITMA**  
**SEMESTER II 2022-2023**

**Pencarian Pasangan Titik Terdekat**  
**dengan Algoritma *Divide and Conquer***

Disusun oleh:

Bintang Dwi Marthen

13521144



**PROGRAM STUDI**  
**TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO**  
**DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG 2022**

## 1. ALGORITMA

### 1.1. Algoritma *Brute Force*

1. Jarak minimal ditetapkan sebagai tak terbatas
2. Iterasikan setiap pasangan titik dan cari jaraknya
3. Apabila jarak antar titik lebih rendah dari jarak minimal, maka catat jarak minimal sebagai jarak antar titik tersebut

### 1.2. Algoritma *Quick Sort*

1. Nilai pertama dipilih sebagai *pivot*
2. Cari jumlah nilai yang lebih kecil dari *pivot*
3. Tempatkan *pivot* pada posisinya berdasarkan nilai yang telah didapatkan pada langkah 2
4. Periksa nilai-nilai di kiri dan kanan *pivot* dan lakukan pertukaran hingga semua nilai di kiri *pivot* lebih kecil dari *pivot* dan semua nilai di kanan *pivot* lebih besar dari *pivot*
5. Lakukan langkah 1 – 4 untuk nilai-nilai di kiri dan kanan *pivot* hingga indeks paling kiri lebih dari sama dengan indeks paling kanan

### 1.3. Algoritma *Divide and Conquer*

1. Seluruh titik diurutkan menggunakan *Quick Sort* berdasarkan koordinat pertamanya (sumbu-x bila dimensi 2 atau 3)
2. Seluruh titik dibagi menjadi dua berdasarkan dimensi pertamanya (sumbu-x bila dimensi 2 atau 3)
3. Jarak terdekat di kedua sisi akan dicari
4. Jarak yang lebih kecil diantara kedua sisi akan dicatat sebagai jarak minimal
5. Seluruh titik yang berada pada jarak kurang dari jarak minimal dari sumbu dihimpun dalam sebuah senarai
6. Jarak antar titik dalam senarai tersebut akan dicari, bila lebih kecil dari jarak minimal maka akan dicatat sebagai jarak minimal
7. Langkah 2-6 dilakukan secara rekursif hingga jumlah titik kurang dari batas tertentu (dimasukkan oleh pengguna)

### 1.4. Hal-Hal yang Harus Diperhatikan

1. Tidak terdapat penanganan kasus apabila terdapat beberapa pasangan titik yang jaraknya sama dan jaraknya adalah minimal sehingga koordinat-koordinat titik yang ditampilkan oleh algoritma *brute force* dan algoritma *divide and conquer* dapat berbeda
2. Jarak dari algoritma *brute force* diasumsikan selalu benar
3. Terdapat batasan kordinat yaitu 0-1000
4. Terdapat batasan jumlah titik yaitu 2-5000
5. Terdapat batasan dimensi yaitu 1-100
6. Lambatnya algoritma *divide and conquer* dibandingkan dengan *brute force* dapat disebabkan oleh penentuan *bound* yang buruk dan pemanggilan rekursif (pada kasus dengan jumlah titik sedikit)
7. Program akan langsung menggunakan algoritma *brute force* bila dimensi adalah satu dan jumlah titik sudah di bawah *bound*
8. Terdapat program *find.cpp* (dalam implementasinya batasan-batasan dalam tugas kecil ini tidak terlalu diperhatikan karena fungsinya yang hanya sebagai utilitas sampingan) untuk mencari nilai *bound* yang dianggap optimal
9. *Random Number Generator* yang digunakan akan membangkitkan bilangan bulat

## 2. SOURCE CODE DALAM BAHASA C++

### 2.1. Berkas point.hpp

```
#ifndef _POINT_HPP_
#define _POINT_HPP_

#include <vector>
#include <cmath>
using namespace std;

class Point {
private:
    int id;
    int dimension;
    vector<double> coordinates;

public:
    /**
     * @brief Construct a new Point object with default value 0 for
id and dimension
     */
    Point();

    /**
     * @brief Construct a new Point object
     *
     * @param id The id of the point
     * @param dimension The dimension of the point
     * @param coordinates The coordinates of the point
     */
    Point(int id, int dimension, vector<double> coordinates);

    /**
     * @brief Construct a new Point object from another point
     */
    Point(const Point& p);

    /**
     * @brief Assign a point to another point
     */
    void operator=(const Point& p);

    /**
     * @brief Get the Id of the point
     */
    int getId();

    /**
     * @brief Get the Dimension of the point
     */
    int getDimension();
};
```

```
/**
 * @brief Get the Coordinate of the point in the i-th dimension
 *
 */
double getCoordinate(int i);

/**
 * @brief The euclidean distance this point and point p
 *
 */
double distance(const Point& p) const;

/**
 * @brief Print the coordinates of the point
 *
 */
void print();
};

#endif
```

## 2.2. Berkas point.cpp

```
#include "header/point.hpp"
#include <cmath>
#include <iostream>

Point::Point() {
    id = 0;
    dimension = 0;
    coordinates = vector<double>();
}

Point::Point(int id, int dimension, vector<double> coordinates){
    this->id = id;
    this->dimension = dimension;
    this->coordinates = coordinates;
}

Point::Point(const Point& p){
    id = p.id;
    dimension = p.dimension;
    coordinates = p.coordinates;
}

void Point::operator=(const Point& p) {
    id = p.id;
    dimension = p.dimension;
    coordinates = p.coordinates;
}

int Point::getId(){
    return id;
}

int Point::getDimension() {
    return dimension;
}
```

```
}

double Point::getCoordinate(int i) {
    return coordinates[i];
}

double Point::distance(const Point& p) const {
    double sum = 0;
    for (int i = 0; i < dimension; i++) {
        sum += pow(coordinates[i] - p.coordinates[i], 2);
    }
    return sqrt(sum);
}

void Point::print(){
    cout<<"(";
    for(int i = 0; i < dimension; i++){
        cout<<coordinates[i];
        if(i != dimension - 1){
            cout<<" ";
        }
    }
    cout<<")"<<endl;
}
```

### 2.3. Berkas io.hpp

```
#ifndef _IO_HPP_
#define _IO_HPP_

#include "point.hpp"
#include <iostream>
#include <vector>
using namespace std;

/**
 * @brief Validate whether the input is a number between min and max
 */
int validate(int min, int max);

/**
 * @brief Generate n random points in the dimension
 */
vector<Point> randomPoints(int dimension, int n);

/**
 * @brief Read n points from the standard input
 */
vector<Point> readPoints(int dimension, int n);

/**
 * @brief Read n points from the file with assumption that the file is in
the format
 */
```

```
*/  
vector<Point> readPoints(string filename);  
  
/**  
 * @brief Plot the points and the closest pair  
 */  
void plotPoints(vector<Point> points, pair<Point, Point> closestPair);  
#endif
```

## 2.4. Berkas io.cpp

```
#include "header/io.hpp"  
#include <sstream>  
#include <fstream>  
  
double random(int limit = 1000){  
    unsigned long long x;  
    __builtin_ia32_rdrand64_step(&x);  
    return (double) (x % (int) (limit+1));  
}  
  
int validate(int min, int max){  
    int x;  
    while(!(cin >> x) || x < min || x > max){  
        cin.clear();  
        cin.ignore(1000, '\n');  
        cout << "Invalid input. Please input an integer between " << min  
<< " and " << max << ": ";  
    }  
    cin.clear();  
    cin.ignore(1000, '\n');  
    return x;  
}  
  
bool validate(double x){  
    return x >= 0 && x <= 1000;  
}  
  
bool validate(string input, int dimension){  
    /*  
        cin.clear() and cin.ignore() is used to clear the input buffer  
    */  
  
    stringstream ss(input);  
    string coordinate;  
    int count = 0;  
    while(ss >> coordinate){  
        double x;  
        try{  
            x = stod(coordinate);  
        } catch (const std::invalid_argument& ia) {  
            cin.clear();  
            cin.ignore(1000, '\n');  
            return false;  
        }  
    }  
}
```

```

        if(!validate(x)){
            cin.clear();
            cin.ignore(1000, '\n');
            return false;
        }
        count++;
    }
    if(count != dimension){
        return false;
    }
    cin.clear();
    cin.ignore(1000, '\n');
    return true;
}

vector<Point> randomPoints(int dimension, int n){
    vector<Point> points;
    for(int i = 0; i < n; i++){
        vector<double> coordinates;
        for(int j = 0; j < dimension; j++){
            double x = random();
            coordinates.push_back(x);
        }
        Point p(i, dimension, coordinates);
        points.push_back(p);
    }
    return points;
}

vector<Point> readPoints(int dimension, int n){
    cout << "Input " << n << " points in " << dimension << " dimension"
    << endl;
    vector<Point> points;
    for(int i = 0; i < n; i++){
        cout << "Point " << i+1 << ": ";
        string input;
        getline(cin, input);
        while(!validate(input, dimension)){
            cout << "Invalid input" << endl;
            cout << "Point " << i+1 << ": ";
            getline(cin, input);
        }
        stringstream ss(input);
        string coordinate;
        vector<double> coordinates;
        while(ss >> coordinate){
            double x = stod(coordinate);
            coordinates.push_back(x);
        }
        Point p(i, dimension, coordinates);
        points.push_back(p);
    }
    return points;
}

vector<Point> readPoints(string filename){

```

```

    ifstream file(filename);
    string line;
    vector<Point> points;
    int dimension;
    int n = 0;
    while(getline(file, line)){
        stringstream ss(line);
        string coordinate;
        vector<double> coordinates;
        int count = 0;
        while(ss >> coordinate){
            double x;
            try{
                x = stod(coordinate);
            } catch (const std::invalid_argument& ia) {
                cout << "Invalid input" << endl;
                exit(1);
            }
            if(!validate(x)){
                cout << "Invalid input" << endl;
                exit(1);
            }
            coordinates.push_back(x);
            count++;
        }
        if(n == 0){
            dimension = count;
        }
        if(count != dimension){
            cout << "Invalid input" << endl;
            exit(1);
        }
        Point p(n, dimension, coordinates);
        points.push_back(p);
        n++;
        if(n > 5000){
            cout << "Too many points" << endl;
            cout << "Please input less than 5000 points" << endl;
            exit(1);
        }
    }
    return points;
}

void plotPoints(vector<Point> points, pair<Point, Point> closestPair){
    cout << "Plotting the points and the closest pair" << endl;
    ofstream file("test/plot.py");
    file << "import matplotlib.pyplot as plt" << endl;
    file << "import numpy as np" << endl;
    file << "import math" << endl;
    file << "x = []" << endl;
    file << "y = []" << endl;
    if(points[0].getDimension() == 3){
        file << "z = []" << endl;
        for(int i = 0; i < points.size(); i++){

```



```

        if(points[i].getId() != closestPair.first.getId() &&
points[i].getId() != closestPair.second.getId()){
            file << "x.append(" << points[i].getCoordinate(0) << ")"
<< endl;
            file << "y.append(" << points[i].getCoordinate(1) << ")"
<< endl;
            file << "z.append(" << points[i].getCoordinate(2) << ")"
<< endl;
        }
    }
    file << "fig = plt.figure()" << endl;
    file << "ax = fig.add_subplot(111, projection='3d')" << endl;
    file << "ax.scatter(x, y, z)" << endl;
    file << "ax.scatter([" << closestPair.first.getCoordinate(0) <<
", " << closestPair.second.getCoordinate(0) << "], [" <<
closestPair.first.getCoordinate(1) << " ", " <<
closestPair.second.getCoordinate(1) << "], [" <<
closestPair.first.getCoordinate(2) << " ", " <<
closestPair.second.getCoordinate(2) << "], color='red')" << endl;
}
    if(points[0].getDimension() == 2){
        for(int i = 0; i < points.size(); i++){
            if(points[i].getId() != closestPair.first.getId() &&
points[i].getId() != closestPair.second.getId()){
                file << "x.append(" << points[i].getCoordinate(0) << ")"
<< endl;
                file << "y.append(" << points[i].getCoordinate(1) << ")"
<< endl;
            }
        }
        file << "plt.scatter(x, y)" << endl;
        file << "plt.scatter([" << closestPair.first.getCoordinate(0) <<
", " << closestPair.second.getCoordinate(0) << "], [" <<
closestPair.first.getCoordinate(1) << " ", " <<
closestPair.second.getCoordinate(1) << "], color='red')" << endl;
    }
    file << "plt.savefig('test/plot.png')" << endl;
    file.close();
    system("python test/plot.py");
    cout << "The points are plotted" << endl;
}

```

## 2.5. Berkas solver.hpp

```

#ifndef _SOLVER_HPP_
#define _SOLVER_HPP_

#include "point.hpp"
#include <vector>
#include <utility>
using namespace std;

/**
 * @brief Sort the points based on the coordinate of the (k+1)-th dimension
 *
 */
void qSort(vector<Point>& points, int left, int right, int k);

```

```
/**
 * @brief Find the closest pair of points using brute force
 *
 * @param points Vector of points
 * @param count Number of distance calculation
 * @return pair<Point, Point> of the closest pair of points
 */
pair<Point, Point> bruteForce(vector<Point>& points, int& count);

/**
 * @brief Find the closest pair of points using divide and conquer
 *
 * @param points Vector of points
 * @param count Number of distance calculation
 * @param bound The boundary of number of points before using brute force
 * @return pair<Point, Point> of the closest pair of points
 */
pair<Point, Point> divideConquer(vector<Point>& points, int& count, int
bound);

#endif
```

## 2.6. Berkas solver.cpp

```
#include "header/solver.hpp"
#include <cmath>

void swap(vector<Point>& points, int i, int j){
    Point temp = points[i];
    points[i] = points[j];
    points[j] = temp;
}

double min(double a, double b){
    return a < b ? a : b;
}

int partition(vector<Point>& points, int left, int right, int k){
    int pivot = left;
    int count = 0;

    for(int i = left + 1; i <= right; i++){
        if(points[i].getCoordinate(k) < points[pivot].getCoordinate(k)){
            count++;
        }
    }

    swap(points, pivot, left + count);

    int i = left, j = right;
    int idx = left + count;

    while(i < idx && j > idx){
        if(points[i].getCoordinate(k) < points[idx].getCoordinate(k)){
            i++;
        }
        if(points[j].getCoordinate(k) > points[idx].getCoordinate(k)){
            j--;
        }
    }
    swap(points, i, j);
}
```

```

    } else if(points[j].getCoordinate(k) >
points[idx].getCoordinate(k)){
    j--;
    } else {
    swap(points, i, j);
    i++;
    j--;
    }
  }

  return idx;
}

void qSort(vector<Point>& points, int left, int right, int k){
  if(left >= right){
    return;
  }

  int pivot = partition(points, left, right, k);

  qSort(points, left, pivot - 1, k);
  qSort(points, pivot + 1, right, k);
}

pair<Point, Point> bruteForce(vector<Point>& points, int& count){
  double minDistance = DBL_MAX;
  Point p1, p2;

  for(int i = 0; i < points.size(); i++){
    for(int j = i + 1; j < points.size(); j++){
      count++;
      double distance = points[i].distance(points[j]);
      if(distance < minDistance){
        minDistance = distance;
        p1 = points[i];
        p2 = points[j];
      }
    }
  }

  return make_pair(p1, p2);
}

pair<Point, Point> divideUtil(vector<Point>& points, int& count, int
bound){
  if(points.size() <= bound || points[0].getDimension() == 1){
    return bruteForce(points, count);
  }

  int mid = points.size() / 2;
  vector<Point> left(points.begin(), points.begin() + mid);
  vector<Point> right(points.begin() + mid, points.end());

  pair<Point, Point> leftPair = divideUtil(left, count, bound);
  pair<Point, Point> rightPair = divideUtil(right, count, bound);

```

```

    Point p1, p2;
    double minDistance;
    if(leftPair.first.distance(leftPair.second) <
rightPair.first.distance(rightPair.second)){
        p1 = leftPair.first;
        p2 = leftPair.second;
        minDistance = leftPair.first.distance(leftPair.second);
    } else {
        p1 = rightPair.first;
        p2 = rightPair.second;
        minDistance = rightPair.first.distance(rightPair.second);
    }

    vector<Point> strip;
    for(int i = 0; i < points.size(); i++){
        if(abs(points[i].getCoordinate(0) - points[mid].getCoordinate(0))
< minDistance){
            strip.push_back(points[i]);
        }
    }

    // qSort(strip, 0, strip.size() - 1, 1);

    for(int i = 0; i < strip.size(); i++){
        //for(int j = i + 1; j < strip.size() && (strip[j].getCoordinate(1)
- strip[i].getCoordinate(1)) < minDistance; j++){
        for(int j = i + 1; j < strip.size(); j++){
            count++;
            double distance = strip[i].distance(strip[j]);
            if(distance < minDistance){
                minDistance = distance;
                p1 = strip[i];
                p2 = strip[j];
            }
        }
    }
    return(make_pair(p1, p2));
}

pair<Point, Point> divideConquer(vector<Point>& points, int& count, int
bound){
    qSort(points, 0, points.size() - 1, 1);
    return divideUtil(points, count, bound);
}

```

## 2.7. Berkas find.cpp

```

#include "lib/header/io.hpp"
#include "lib/header/solver.hpp"
#include <chrono>
#include <numeric>
#include <algorithm>

int main(){
    int dimension;
    int num;
    cout << "Enter dimension: "; cin >> dimension;

```

```

        cout << "Enter number of points: "; cin >> num;
        vector<int> bestBounds;
        vector<Point> points = randomPoints(dimension, num);
        int primeBound;
        int dummy = 0;
        int iterations;
        cout << "Enter number of iterations: "; cin >> iterations;
        pair<Point, Point> dummyPair;
        for(int k = 0; k < iterations; k++){
            // cout << "Iteration: " << k+1 << endl;
            auto primeDuration = chrono::duration<double, milli>(1000000000).count();
            for(int i = 3; i < 100; i++){
                // cout << "Bound: " << i << endl;
                auto start = chrono::high_resolution_clock::now();
                dummyPair = bruteForce(points, dummy);
                dummy = 0;
                auto end = chrono::high_resolution_clock::now();
                auto duration = chrono::duration<double, milli>(end - start).count();
                // cout << "BruteForce: " << duration << " ms" << endl;
                auto start2 = chrono::high_resolution_clock::now();
                dummyPair = divideConquer(points, dummy, i);
                dummy = 0;
                auto end2 = chrono::high_resolution_clock::now();
                auto duration2 = chrono::duration<double, milli>(end2 - start2).count();
                // cout << "DivideConquer: " << duration2 << " ms" << endl;
                if(duration2 < duration && (duration2 - duration) < primeDuration){
                    primeBound = i;
                    primeDuration = duration2 - duration;
                }
                // cout << "-----\n";
            }
            // cout << "Best :" << primeBound << endl;
            bestBounds.push_back(primeBound);
        }
        double avg = accumulate(bestBounds.begin(), bestBounds.end(), 0.0) / bestBounds.size();
        int count[10] = {0};
        for(auto i : bestBounds){
            count[i/10]++;
        }
        for(int i = 0 ; i< 10; i++){
            cout << i*10 << " - " << (i+1)*10 << ": " << count[i] << endl;
        }
        cout << "Average: " << avg << endl;
        sort(bestBounds.begin(), bestBounds.end());
        cout << "Median: " << bestBounds[iterations/2] << endl;
        return 0;
    }

```

## 2.8. Berkas main.cpp

```

#include "lib/header/io.hpp"
#include "lib/header/solver.hpp"

```

```

#include <chrono>

int main(){
    int n;
    cout<<"Input options:\n 1. Random points\n 2. Read from file\n 3. Read
from standard input\n";
    cout<<"Enter your choice: "; int ans = validate(1,3);
    vector<Point> points;
    string filename;
    int dimension;
    switch(ans){
        case 1:
            cout<<"Enter    number    of    points(2-5000):    ";    n    =
validate(2,5000);
            cout<<"Enter    dimension(1-100):    ";    dimension    =    validate(1,
100);
            points = randomPoints(dimension, n);
            break;
        case 2:
            cout<<"Enter file name: "; cin>>filename;
            points = readPoints(filename);
            break;
        case 3:
            cout<<"Enter    number    of    points(2-5000):    ";    n    =
validate(2,5000);
            cout<<"Enter    dimension(1-100):    ";    dimension    =    validate(1,
100);
            points = readPoints(dimension, n);
            break;
        default:
            cout<<"Invalid input"<<endl;
            exit(1);
    }

    cout<<"Enter number of points before using brute force (min 3): "; int
bound = validate(3, INT_MAX);

    int count = 0;
    auto start = chrono::high_resolution_clock::now();
    pair<Point, Point> closestPair = bruteForce(points, count);
    auto end = chrono::high_resolution_clock::now();
    auto duration = chrono::duration<double, milli>(end - start).count();
    cout << "Number of operations: " << count << endl;
    cout << "Closest pair of points: " << closestPair.first.getId() << "
and " << closestPair.second.getId() << endl;
    cout << "Coordinate of the first point: "; closestPair.first.print();
    cout << "Coordinate    of    the    second    point:    ";
closestPair.second.print();
    cout << "Distance: " <<
closestPair.first.distance(closestPair.second) << endl;
    cout << duration << " ms" << endl;

    count = 0;
    start = chrono::high_resolution_clock::now();
    closestPair = divideConquer(points, count, bound);
    end = chrono::high_resolution_clock::now();
  
```

```

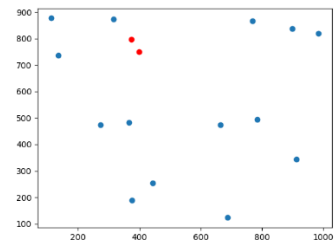
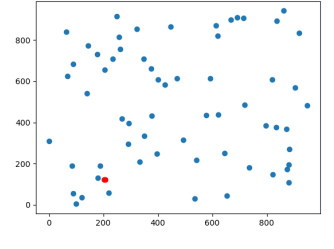
duration = chrono::duration<double, milli>(end - start).count();
cout << "Number of operations: " << count << endl;
cout << "Closest pair of points: " << closestPair.first.getId() << "
and " << closestPair.second.getId() << endl;
cout << "Coordinate of the first point: "; closestPair.first.print();
cout << "Coordinate of the second point: ";
closestPair.second.print();
cout << "Distance: " <<
closestPair.first.distance(closestPair.second) << endl;
cout << duration << " ms" << endl;

cout << "Ran on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB
of RAMs\n";

if(points[0].getDimension() == 2 || points[0].getDimension() == 3){
    cout << "Do you want to plot the points?\n 1. Yes\n 2. No\n";
    cout << "Enter your choice: "; ans = validate(1,2);
    if(ans == 1){
        plotPoints(points, closestPair);
    }
}
return 0;
}
    
```

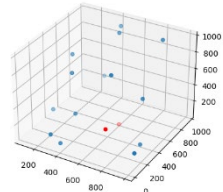
### 3. HASIL EKSEKUSI PROGRAM

#### 3.1. Dimensi Dua (Jumlah Titik = 16, 64, 128, 1000)

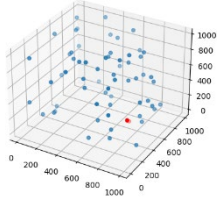
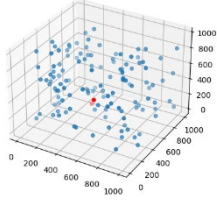
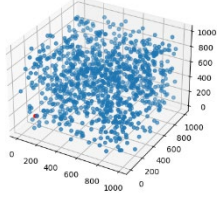
Tangkapan Layar Find	Tangkapan Layar Main	Hasil Plot
Tidak ada karena n terlalu kecil (bila digunakan maka langsung <i>brute force</i> )	Input options: 1. Random points 2. Read from file 3. Read from standard input Enter your choice: 1 Enter number of points(2-5000): 16 Enter dimension(1-100): 2 Enter number of points before using brute force (min 3): 3 Number of operations: 128 Closest pair of points: 2 and 7 Coordinate of the first point: (374, 796) Coordinate of the second point: (398, 751) Distance: 51 0.0299 ms Number of operations: 19 Closest pair of points: 7 and 2 Coordinate of the first point: (398, 751) Coordinate of the second point: (374, 796) Distance: 51 0.0575 ms Ran on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB of RAMs Do you want to plot the points? 1. Yes 2. No Enter your choice: 1 Plotting the points and the closest pair The points are plotted	
Enter dimension: 2 Enter number of points: 64 Enter number of iterations: 100 0 - 10: 3 10 - 20: 17 20 - 30: 26 30 - 40: 11 40 - 50: 12 50 - 60: 12 60 - 70: 7 70 - 80: 3 80 - 90: 5 90 - 100: 4 Average: 39.71 Median: 34	Input options: 1. Random points 2. Read from file 3. Read from standard input Enter your choice: 1 Enter number of points(2-5000): 64 Enter dimension(1-100): 2 Enter number of points before using brute force (min 3): 34 Number of operations: 2016 Closest pair of points: 1 and 31 Coordinate of the first point: (284, 121) Coordinate of the second point: (207, 121) Distance: 3 0.1694 ms Number of operations: 992 Closest pair of points: 1 and 31 Coordinate of the first point: (284, 121) Coordinate of the second point: (207, 121) Distance: 3 0.1499 ms Ran on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB of RAMs Do you want to plot the points? 1. Yes 2. No Enter your choice: 1 Plotting the points and the closest pair The points are plotted	

<pre> Enter dimension: 2 Enter number of points: 128 Enter number of iterations: 100 0 - 10: 7 10 - 20: 12 20 - 30: 12 30 - 40: 15 40 - 50: 10 50 - 60: 9 60 - 70: 9 70 - 80: 8 80 - 90: 11 90 - 100: 7 Average: 47.14 Median: 45                     </pre>	<pre> Input options: 1. Random points 2. Read from file 3. Read from standard input Enter your choice: 1 Enter number of points(2-5000): 128 Enter dimension(1-100): 2 Enter number of points before using brute force (min 3): 45 Number of operations: 8128 Closest pair of points: 123 and 125 Coordinate of the first point: (995, 865) Coordinate of the second point: (994, 867) Distance: 2.23607 0.472 ms Number of operations: 1987 Closest pair of points: 123 and 125 Coordinate of the first point: (995, 865) Coordinate of the second point: (994, 867) Distance: 2.23607 0.2824 ms Ran on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB of RAMs Do you want to plot the points? 1. Yes 2. No Enter your choice: 1 Plotting the points and the closest pair The points are plotted                     </pre>	
<pre> Enter dimension: 2 Enter number of points: 1000 Enter number of iterations: 100 0 - 10: 5 10 - 20: 14 20 - 30: 16 30 - 40: 11 40 - 50: 10 50 - 60: 8 60 - 70: 12 70 - 80: 8 80 - 90: 5 90 - 100: 11 Average: 47.58 Median: 45                     </pre>	<pre> Input options: 1. Random points 2. Read from file 3. Read from standard input Enter your choice: 1 Enter number of points(2-5000): 1000 Enter dimension(1-100): 2 Enter number of points before using brute force (min 3): 45 Number of operations: 499500 Closest pair of points: 96 and 756 Coordinate of the first point: (106, 382) Coordinate of the second point: (106, 382) Distance: 0 28.985 ms Number of operations: 15150 Closest pair of points: 632 and 598 Coordinate of the first point: (608, 705) Coordinate of the second point: (608, 705) Distance: 0 2.9653 ms Ran on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB of RAMs Do you want to plot the points? 1. Yes 2. No Enter your choice: 1 Plotting the points and the closest pair The points are plotted                     </pre>	

### 3.2. Dimensi Tiga (Jumlah Titik = 16, 64, 128, 1000)

Tangkapan Layar Find	Tangkapan Layar Main	Hasil Plot
<p>Tidak ada karena n terlalu kecil (bila digunakan maka langsung <i>brute force</i>)</p>	<pre> Input options: 1. Random points 2. Read from file 3. Read from standard input Enter your choice: 1 Enter number of points(2-5000): 16 Enter dimension(1-100): 3 Enter number of points before using brute force (min 3): 3 Number of operations: 120 Closest pair of points: 3 and 6 Coordinate of the first point: (774, 268, 310) Coordinate of the second point: (697, 193, 268) Distance: 115.404 0.0313 ms Number of operations: 31 Closest pair of points: 6 and 3 Coordinate of the first point: (697, 193, 268) Coordinate of the second point: (774, 268, 310) Distance: 115.404 0.0556 ms Ran on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB of RAMs Do you want to plot the points? 1. Yes 2. No Enter your choice: 1 Plotting the points and the closest pair The points are plotted                     </pre>	



<pre> Enter dimension: 3 Enter number of points: 64 Enter number of iterations: 100 0 - 10: 4 10 - 20: 16 20 - 30: 12 30 - 40: 15 40 - 50: 9 50 - 60: 17 60 - 70: 12 70 - 80: 6 80 - 90: 5 90 - 100: 4 Average: 43.77 Median: 44 </pre>	<pre> Input options: 1. Random points 2. Read from file 3. Read from standard input Enter your choice: 1 Enter number of points(2-5000): 64 Enter dimension(1-100): 3 Enter number of points before using brute force (min 3): 44 Number of operations: 2816 Closest pair of points: 31 and 58 Coordinate of the first point: (915, 234, 368) Coordinate of the second point: (909, 211, 394) Distance: 35.2278 0.2899 ms Number of operations: 995 Closest pair of points: 58 and 31 Coordinate of the first point: (909, 211, 394) Coordinate of the second point: (915, 234, 368) Distance: 35.2278 0.1613 ms Ran on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB of RAMs Do you want to plot the points? 1. Yes 2. No Enter your choice: 1 Plotting the points and the closest pair The points are plotted </pre>	
<pre> Enter dimension: 3 Enter number of points: 128 Enter number of iterations: 100 0 - 10: 14 10 - 20: 16 20 - 30: 11 30 - 40: 15 40 - 50: 7 50 - 60: 8 60 - 70: 9 70 - 80: 5 80 - 90: 7 90 - 100: 8 Average: 41.48 Median: 36 </pre>	<pre> Input options: 1. Random points 2. Read from file 3. Read from standard input Enter your choice: 1 Enter number of points(2-5000): 128 Enter dimension(1-100): 3 Enter number of points before using brute force (min 3): 41 Number of operations: 8128 Closest pair of points: 23 and 76 Coordinate of the first point: (664, 112, 610) Coordinate of the second point: (649, 102, 563) Distance: 50.3389 1.2364 ms Number of operations: 2113 Closest pair of points: 76 and 23 Coordinate of the first point: (649, 102, 563) Coordinate of the second point: (664, 112, 610) Distance: 50.3389 0.3342 ms Ran on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB of RAMs Do you want to plot the points? 1. Yes 2. No Enter your choice: 1 Plotting the points and the closest pair The points are plotted </pre>	
<pre> Enter dimension: 3 Enter number of points: 1000 Enter number of iterations: 100 0 - 10: 8 10 - 20: 15 20 - 30: 11 30 - 40: 16 40 - 50: 7 50 - 60: 16 60 - 70: 5 70 - 80: 5 80 - 90: 6 90 - 100: 11 Average: 45.27 Median: 41 </pre>	<pre> Input options: 1. Random points 2. Read from file 3. Read from standard input Enter your choice: 1 Enter number of points(2-5000): 1000 Enter dimension(1-100): 3 Enter number of points before using brute force (min 3): 41 Number of operations: 499500 Closest pair of points: 216 and 531 Coordinate of the first point: (32, 164, 71) Coordinate of the second point: (38, 168, 77) Distance: 9.38083 37.9263 ms Number of operations: 15763 Closest pair of points: 216 and 531 Coordinate of the first point: (32, 164, 71) Coordinate of the second point: (38, 168, 77) Distance: 9.38083 3.4998 ms Ran on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB of RAMs Do you want to plot the points? 1. Yes 2. No Enter your choice: 1 Plotting the points and the closest pair The points are plotted </pre>	

### 3.3. Dimensi Lebih dari Tiga

(Dimensi, Jumlah Titik)	Tangkapan Layar Find	Tangkapan Layar Main
1 dimensi 1000 titik	Tidak ada karena hanya satu dimensi langsung ke <i>brute force</i>	<pre> Input options: 1. Random points 2. Read from file 3. Read from standard input Enter your choice: 1 Enter number of points(2-5000): 1000 Enter dimension(1-100): 1 Enter number of points before using brute force (min 3): 3 Number of operations: 499500 Closest pair of points: 0 and 53 Coordinate of the first point: (818) Coordinate of the second point: (818) Distance: 0 18.2551 ms Number of operations: 499500 Closest pair of points: 643 and 606 Coordinate of the first point: (869) Coordinate of the second point: (869) Distance: 0 19.0324 ms Run on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB of RAMs                     </pre>
5 dimensi 100 titik	<pre> Enter dimension: 5 Enter number of points: 100 Enter number of iterations: 100 0 - 10: 8 10 - 20: 9 20 - 30: 14 30 - 40: 16 40 - 50: 11 50 - 60: 12 60 - 70: 2 70 - 80: 9 80 - 90: 8 90 - 100: 11 Average: 47.38 Median: 43                     </pre>	<pre> Input options: 1. Random points 2. Read from file 3. Read from standard input Enter your choice: 1 Enter number of points(2-5000): 100 Enter dimension(1-100): 5 Enter number of points before using brute force (min 3): 43 Number of operations: 4950 Closest pair of points: 80 and 91 Coordinate of the first point: (7, 28, 68, 291, 230) Coordinate of the second point: (149, 20, 3, 207, 223) Distance: 177.646 1.1943 ms Number of operations: 1726 Closest pair of points: 91 and 80 Coordinate of the first point: (149, 20, 3, 207, 223) Coordinate of the second point: (7, 28, 68, 291, 230) Distance: 177.646 0.4131 ms Run on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB of RAMs                     </pre>
10 dimensi 100 titik	<pre> Enter dimension: 10 Enter number of points: 100 Enter number of iterations: 100 0 - 10: 1 10 - 20: 1 20 - 30: 8 30 - 40: 6 40 - 50: 10 50 - 60: 16 60 - 70: 12 70 - 80: 17 80 - 90: 16 90 - 100: 13 Average: 63.26 Median: 69                     </pre>	<pre> Input options: 1. Random points 2. Read from file 3. Read from standard input Enter your choice: 1 Enter number of points(2-5000): 100 Enter dimension(1-100): 10 Enter number of points before using brute force (min 3): 69 Number of operations: 4950 Closest pair of points: 59 and 78 Coordinate of the first point: (797, 237, 265, 608, 88, 877, 935, 602, 329, 741) Coordinate of the second point: (946, 261, 143, 543, 247, 917, 700, 581, 267, 585) Distance: 390.657 1.1438 ms Number of operations: 3828 Closest pair of points: 59 and 78 Coordinate of the first point: (797, 237, 265, 608, 88, 877, 935, 602, 329, 741) Coordinate of the second point: (946, 261, 143, 543, 247, 917, 700, 581, 267, 585) Distance: 390.657 1.0369 ms Run on AMD Ryzen 5 4500U 6-Core Processor 2.40 GHz with 8 GB of RAMs                     </pre>

## 4. LAMPIRAN

### 4.1. Pranala Repository GitHub

[https://github.com/Marthenn/Tucil2\\_13521144](https://github.com/Marthenn/Tucil2_13521144)

### 4.2. Tabel Ketercapaian Program

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa ada kesalahan	✓	
2. Program berhasil <i>running</i>	✓	

3. Program dapat menerima masukan an menuliskan luaran	✓	
4. Luaran program sudah benar (solusi <i>closest pair</i> benar)	✓	
5. Bonus 1 dikerjakan	✓	
6. Bonus 2 dikerjakan	✓	