Laporan Tugas Kecil 2

IF2211/Strategi Algoritma

Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer

Disusun Oleh:

Muhammad Equilibrie Fajria - 13521047

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2023

1. Algoritma Divide and Conquer

Untuk menyelesaikan persoalan mencari pasangan titik terdekat 3d, digunakan algoritma divide and conquer. Algoritma divide and conquer merupakan algoritma yang membagi persoalan menjadi beberapa upa-persoalan dan menyelesaikan tiap upa-persoalan tersebut dan menggabungkan solusinya. Secara singkat, cara kerja algoritma yang digunakan pada persoalan ini yaitu program akan mengurutkan list titik, lalu list titik tersebut dibagi menjadi dua bagian. Setiap bagian akan menghasilkan pasangan terdekat dengan jarak dl dan dr. Lalu, akan dicari titik yang berada di daerah strip berdasarkan jarak terkecil dari perbandingan dl dan dr (d). Setelah itu, akan dicari pasangan terdekat pada list strip (ds) dan akan dibandingkan jaraknya dengan d. Jika d lebih kecil, maka pasangan titik dengan jarak d merupakan pasangan terdekat sedangkan jika sebaliknya, maka pasangan dengan jarak ds merupakan pasangan terdsekat. Program secara rekursif membagi dan menyelesaikan tiap upa-permasalahan. Berikut adalah penjelasan alur program utamanya:

1. Program akan meminta masukan dari pengguna terkait banyak titik (n), banyak dimensi, dan range maksimal dari masukan.
2. Setelah mendapatkan input, program akan mengenerate titik-titik sesuai masukan pengguna.
3. Program lalu mencari solusi dengan pendekatan bruteForce. BruteForce digunakan sebagai pembanding untuk pendekatan divide and conquer. Setelah menjalankan bruteForce, program mencari solusi dengan menggunakan pendekatan divide and conquer.

Berikut penjelasan alur program brute force:

1. Program menghitung shortestDistance menggunakan titik pada indeks ke-0 dan ke-1 pada list.
2. Program melakukan looping untuk mencari pasangan titik terdekat dengan mencari jarak Euclidean antar seluruh titik pada list.

Berikut penjelasan alur program divide and conquer

1. Program mengurutkan list berdasarkan absis dari yang paling kecil ke yang paling besar.
2. Program secara rekursif membelah list menjadi dua bagian dan menyelesaikannya sesuai penjelasan di atas.

2. Kode Program

2.1. main.cpp

```cpp
#include "Utility.hpp"
#include "BruteForce.hpp"
#include "DivideConquer.hpp"
using namespace std;
using namespace std::chrono;

int main() {
    int n;
    int dimensi = 3;
    int maxNum = 100;
    srand(time(NULL));
    vector<double>* listVector;

    cout << "Masukkan banyak titik" << endl;
    cin >> n;
    while (n < 2) {
        cout << "Masukan salah, silahkan masukkan kembali" << endl;
        cin >> n;
    }

    cout << "Masukkan banyak dimensi" << endl;
    cin >> dimensi;
    while (dimensi < 2) {
        cout << "Masukan salah, silahkan masukkan kembali" << endl;
        cin >> dimensi;
    }

    cout << "Masukkan batas ujung (batas ujung diasumsikan simetrik)" << endl;
    cin >> maxNum;
    cout << endl;

    listVector = new vector<double>[n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < dimensi; j++) {
            listVector[i].push_back(getRandomNumber(maxNum));
        }
    }


    auto start1 = high_resolution_clock::now();
    BruteForce bruteForce(listVector, n);
    auto stop1 = high_resolution_clock::now();
    auto duration1 = duration_cast<microseconds>(stop1 - start1);

    bruteForce.print();
```

```
    cout << "Waktu eksekusi = " << duration1.count() << " microseconds" <<
endl;
    cout << endl;

    auto start2 = high_resolution_clock::now();
    DivideConquer divideConquer(listVector, n);
    auto stop2 = high_resolution_clock::now();
    auto duration2 = duration_cast<microseconds>(stop2 - start2);

    divideConquer.print();
    cout << "Waktu eksekusi = " << duration2.count() << " microseconds" <<
endl;
    cout << endl;

    return 0;
}
```

2.2. Utility.hpp

```
#ifndef __UTILITY_HPP__
#define __UTILITY_HPP__

#include <cstdlib>
#include <ctime>
#include <vector>
#include <cmath>
#include <tuple>
#include <iostream>
#include <chrono>
using namespace std;


double getRandomNumber (int maxNum);
// maxNum merupakan nilai mutlak dari batas paling ujung (Asumsi niali mutlak
kedua ujung bernilai sama)
// Mengembalikan satu angka yang random

double calculateEuclidean (vector<double> p1, vector<double> p2);
// Mengembalikan nilai euclidean dari dua buah titik

void sortList (vector<double>* list, int size, int plane);
// Mengembalikan list yang telah terurut membesar berdarkan plane (Menggunakan
quicksort)

int partitionList (vector<double>* list, int start, int end, int plane);
// Mempartisi list dan mengembalikan index pivot

void quickSort (vector<double>* list, int start, int end, int plane);
```

```
// Secara rekursif memanggil partitionList dan mengurtkan elemen

#endif
```

2.3. Utility.cpp

```
#include "Utility.hpp"


double getRandomNumber (int maxNum) {
    double random_num;
    int random_operand1, random_operand2, random_operand3;
    int range1 = 10;
    int range2 = 21;
    int range3 = 10;

    random_operand1 = rand() % range1 + 1;
    random_operand2 = rand() % range2 - 10;
    random_operand3 = rand() % range3 + 11;

    random_num = (random_operand1 * random_operand2 * random_operand3) %
(maxNum);

    return random_num;
}

double calculateEuclidean (vector<double> p1, vector<double> p2) {
    double sum = 0;
    double euclidean;

    for (int i = 0; i < p1.size(); i++) {
        sum += pow((p1.at(i) - p2.at(i)), 2);
    }

    euclidean = sqrt(sum);

    return euclidean;
}

void sortList (vector<double>* list, int size, int plane) {
    quickSort(list, 0, size-1, plane);
}

int partitionList (vector<double>* list, int start, int end, int plane) {
    vector<double> pivot = list[start];
    int count = 0;

    for (int i = start + 1; i <= end; i++) {
```

```
        if (list[i].at(plane) <= pivot.at(plane)) {
            count++;
        }
    }

    int pivotIndex = start + count;
    swap(list[pivotIndex], list[start]);

    int i = start, j = end;
    while (i < pivotIndex && j > pivotIndex) {
        while (list[i].at(plane) <= pivot.at(plane)) {
            i++;
        }
        while (list[j].at(plane) > pivot.at(plane)) {
            j--;
        }
        if (i < pivotIndex && j > pivotIndex) {
            swap(list[i++], list[j--]);
        }
    }

    return pivotIndex;
}

void quickSort(vector<double>* list, int start, int end, int plane) {
    // Basis
    if (start >= end)
        return;

    // Rekurens
        // Mempartisi list
    int pivotIndex = partitionList(list, start, end, plane);
        // Mengurutkan bagian kiri pivot
    quickSort(list, start, pivotIndex - 1, plane);
        // Mengurutkan bagian kanan pivot
    quickSort(list, pivotIndex + 1, end, plane);
}
```

2.4. BruteForce.hpp

```
#ifndef __BruteForce_HPP__
#define __BruteForce_HPP__

#include "Utility.hpp"


class BruteForce {
private:
```

```
    int euclideanCount;
    double shortestDistance;
    std::vector<double> p1;
    std::vector<double> p2;

public:
    // CTOR
    BruteForce();
    BruteForce(std::vector<double>* listVector, int size);
    // DTOR
    ~BruteForce();

    // Getter
    int getEuclideanCount();
    double getShortestDistance();
    std::vector<double> getPoint1();
    std::vector<double> getPoint2();

    // Other
    void print();
    // Menampilkan jarak terdekat, pasangan titik terdekat, dan banyaknya
operasi euclidean
};


#endif
```

2.5. BruteForce.cpp

```
#include "BruteForce.hpp"
using namespace std;

BruteForce::BruteForce() : euclideanCount(0), shortestDistance(9999) {}

BruteForce::BruteForce(std::vector<double>* listVector, int size) {
    this->shortestDistance = calculateEuclidean(listVector[0], listVector[1]);
    this->euclideanCount = 1;
    this->p1 = listVector[0];
    this->p2 = listVector[1];
    double temp_euclidean;

    if (size > 2) {
        for (int i = 0; i < size; i++) {
            for (int j = i + 1; j < size; j++) {
                temp_euclidean = calculateEuclidean(listVector[i],
listVector[j]);
                this->euclideanCount++;
                if (temp_euclidean < this->shortestDistance) {
```

```
                    this->shortestDistance = temp_euclidean;
                    this->p1 = listVector[i];
                    this->p2 = listVector[j];
                }
            }
        }
    }
}

BruteForce::~BruteForce() {}

int BruteForce::getEuclideanCount() {
    return this->euclideanCount;
}

double BruteForce::getShortestDistance() {
    return this->shortestDistance;
}

std::vector<double> BruteForce::getPoint1() {
    return this->p1;
}

std::vector<double> BruteForce::getPoint2() {
    return this->p2;
}

void BruteForce::print() {
    cout << "Hasil menggunakan algoritma brute force: " << endl;
    cout << "Jarak terdekat = " << this->shortestDistance << endl;

    cout << "Titik 1 = {";
    for (int i = 0; i < this->p1.size(); i++) {
        cout << this->p1.at(i);
        if (i != this->p1.size() - 1) {
            cout << ", ";
        }
    }
    cout << "}" << endl;

    cout << "Titik 2 = {";
    for (int i = 0; i < this->p2.size(); i++) {
        cout << this->p2.at(i);
        if (i != this->p2.size() - 1) {
            cout << ", ";
        }
    }
    cout << "}" << endl;
```

```
    cout << "Banyaknya operasi euclidean = " << this->euclideanCount << endl;
}
```

2.6. DivideConquer.hpp

```
#ifndef __DIVIDECONQUER_HPP__
#define __DIVIDECONQUER_HPP__

#include "Utility.hpp"


class DivideConquer {
private:
    int euclideanCount;
    double shortestDistance;
    std::vector<double> p1;
    std::vector<double> p2;

public:
    // CTOR
    DivideConquer();
    DivideConquer(std::vector<double>* listVector, int size);
    // DTOR
    ~DivideConquer();

    // Getter
    int getEuclideanCount();
    double getShortestDistance();
    std::vector<double> getPoint1();
    std::vector<double> getPoint2();

    // Other
    std::tuple<double, int, int> dncShortestEuclidean (vector<double>* list,
int start, int end);
    // Mengembailikan euclidean terpendek dari list

    std::tuple<double, int, int> stripShortestEuclidean (vector<double>*
listVector, int start, int end, double d);
    // Mencari euclidean terpendek dari stripList

    void print();
    // Menampilkan jarak terdekat, pasangan titik terdekat, dan banyaknya
operasi euclidean
};

#endif
```

2.7. DivideConquer.cpp

```
#include "DivideConquer.hpp"
using namespace std;


DivideConquer::DivideConquer() : euclideanCount(0), shortestDistance(9999) {}

DivideConquer::DivideConquer(std::vector<double>* listVector, int size) {
    this->euclideanCount = 0;
    std::tuple<double, int, int> retVal;
    sortList(listVector, size, 0);

    if (size > 2) {
        retVal = dncShortestEuclidean(listVector, 0, size-1);
        this->shortestDistance = get<0>(retVal);
        this->p1 = listVector[get<1>(retVal)];
        this->p2 = listVector[get<2>(retVal)];
    }
    else {
        this->shortestDistance = calculateEuclidean(listVector[0],
listVector[1]);
        this->p1 = listVector[0];
        this->p2 = listVector[1];
        this->euclideanCount++;
    }
}

DivideConquer::~DivideConquer() {}

int DivideConquer::getEuclideanCount() {
    return this->euclideanCount;
}

double DivideConquer::getShortestDistance() {
    return this->shortestDistance;
}

std::vector<double> DivideConquer::getPoint1() {
    return this->p1;
}

std::vector<double> DivideConquer::getPoint2() {
    return this->p2;
}

std::tuple<double, int, int> DivideConquer::dncShortestEuclidean
(vector<double>* list, int start, int end) {
    double middlePoint, d;
```

```
    int leftIndex = (start+end)/2;
    int rightIndex = ((start+end)/2) + 1;
    bool out_of_strip_left = false;
    bool out_of_strip_right = false;
    std::tuple<double, int, int> retVal, strip;

    // Basis
    if ((end-start) <= 2) {
        double temp_euclidean = calculateEuclidean(list[start],
list[start+1]);
        this->euclideanCount++;
        double shortest_euclidean = temp_euclidean;
        retVal = {temp_euclidean, start, start+1};

        if ((end-start) > 1) {
            for (int i = start; i <= end; i++) {
                for (int j = i + 1; j <= end; j++) {
                    temp_euclidean = calculateEuclidean(list[i], list[j]);
                    this->euclideanCount++;
                    if (temp_euclidean < shortest_euclidean) {
                        retVal = {temp_euclidean, i, j};
                    }
                }
            }
        }
        return retVal;
    }
    // Rekurens
    else {
        auto leftPart = dncShortestEuclidean (list, start, leftIndex);
        auto rightPart = dncShortestEuclidean (list, rightIndex, end);

        if (get<0>(leftPart) <= get<0>(rightPart)) {
            retVal = leftPart;
        }
        else {
            retVal = rightPart;
        }

        d = get<0>(retVal);
        middlePoint = (list[leftIndex].at(0) + list[rightIndex].at(0))/2.0;
        while ((leftIndex > start) && (!out_of_strip_left)) {
            if (list[leftIndex].at(0) > (middlePoint - d)) {
                leftIndex--;
            }
            else {
                out_of_strip_left = true;
            }
```

```
        }

        while ((rightIndex < end) && (!out_of_strip_right)) {
            if (list[rightIndex].at(0) < (middlePoint + d)) {
                rightIndex++;
            }
            else {
                out_of_strip_right = true;
            }
        }
        strip = stripShortestEuclidean (list, leftIndex+1, rightIndex-1, d);
        if (get<0>(strip) < d) {
            retVal = strip;
        }
        return retVal;
    }
}

std::tuple<double, int, int>
DivideConquer::stripShortestEuclidean(vector<double>* listVector, int start,
int end, double d) {
    double temp_euclidean = calculateEuclidean(listVector[start],
listVector[start+1]);
    this->euclideanCount++;
    double shortest_euclidean = d;
    std::tuple<double, int, int> retVal = {temp_euclidean, start, start+1};

    if (end - start > 1) {
        for (int i = start; i <= end; i++) {
            for (int j = i + 1; j <= end; j++) {
                temp_euclidean = calculateEuclidean(listVector[i],
listVector[j]);
                this->euclideanCount++;
                if (temp_euclidean < shortest_euclidean) {
                    shortest_euclidean = temp_euclidean;
                    retVal = {temp_euclidean, i, j};
                }
            }
        }
    }

    return retVal;
}

void DivideConquer::print() {
    cout << "Hasil menggunakan algoritma divide and conquer: " << endl;
    cout << "Jarak terdekat = " << this->shortestDistance << endl;
```

```
    cout << "Titik 1 = {";
    for (int i = 0; i < this->p1.size(); i++) {
        cout << this->p1.at(i);
        if (i != this->p1.size() - 1) {
            cout << ", ";
        }
    }
    cout << "}" << endl;

    cout << "Titik 2 = {";
    for (int i = 0; i < this->p2.size(); i++) {
        cout << this->p2.at(i);
        if (i != this->p2.size() - 1) {
            cout << ", ";
        }
    }
    cout << "}" << endl;

    cout << "Banyaknya operasi euclidean = " << this->euclideanCount << endl;
}
```

3. Test Case

Test case menggunakan WSL pada windows, disarankan untuk menggunakan environment linux dikarenakan waktu eksekusi pada linux jauh lebih akurat. Berikut adalah spesifkasi hardware yang digunakan:

- AMD Ryzen 7 6800U (8C / 16T, 2.7 / 4.7GHz, 4MB L2 / 16MB L3
- 16GB Soldered LPDDR5-6400
- 512GB SSD M.2 2242 PCIe 4.0x4 NVMe
- Integrated AMD Radeon 680M Graphics

3.1. Dimensi = 3, n = 16

```
Masukkan banyak titik
16
Masukkan banyak dimensi
3
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 179.956
Titik 1 = {-112, 0, -680}
Titik 2 = {-240, -72, -576}
Banyaknya operasi euclidean = 121
Waktu eksekusi = 874 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 179.956
Titik 1 = {-240, -72, -576}
Titik 2 = {-112, 0, -680}
Banyaknya operasi euclidean = 60
Waktu eksekusi = 29 microseconds
```

```
Masukkan banyak titik
16
Masukkan banyak dimensi
3
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 185.138
Titik 1 = {-368, 360, 216}
Titik 2 = {-420, 324, 42}
Banyaknya operasi euclidean = 121
Waktu eksekusi = 99 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 185.138
Titik 1 = {-420, 324, 42}
Titik 2 = {-368, 360, 216}
Banyaknya operasi euclidean = 42
Waktu eksekusi = 32 microseconds
```

3.2. Dimensi > 3, n = 16

```
Masukkan banyak titik
16
Masukkan banyak dimensi
4
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 267.2
Titik 1 = {-200, 350, -220, 26}
Titik 2 = {-336, 170, -280, 156}
Banyaknya operasi euclidean = 121
Waktu eksekusi = 118 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 267.2
Titik 1 = {-336, 170, -280, 156}
Titik 2 = {-200, 350, -220, 26}
Banyaknya operasi euclidean = 90
Waktu eksekusi = 39 microseconds
```

```
Masukkan banyak titik
16
Masukkan banyak dimensi
5
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 387.555
Titik 1 = {39, -180, -432, 17, -224}
Titik 2 = {34, -152, -171, 280, -114}
Banyaknya operasi euclidean = 121
Waktu eksekusi = 98 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 387.555
Titik 1 = {34, -152, -171, 280, -114}
Titik 2 = {39, -180, -432, 17, -224}
Banyaknya operasi euclidean = 118
Waktu eksekusi = 96 microseconds
```

3.3. Dimensi = 3, n =64

```
Masukkan banyak titik
64
Masukkan banyak dimensi
3
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 128.094
Titik 1 = {648, 152, 800}
Titik 2 = {540, 90, 770}
Banyaknya operasi euclidean = 2017
Waktu eksekusi = 655 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 128.094
Titik 1 = {540, 90, 770}
Titik 2 = {648, 152, 800}
Banyaknya operasi euclidean = 718
Waktu eksekusi = 266 microseconds
```

```
Masukkan banyak titik
64
Masukkan banyak dimensi
3
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 63.9453
Titik 1 = {-44, -153, -38}
Titik 2 = {0, -190, -66}
Banyaknya operasi euclidean = 2017
Waktu eksekusi = 650 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 63.9453
Titik 1 = {-44, -153, -38}
Titik 2 = {0, -190, -66}
Banyaknya operasi euclidean = 476
Waktu eksekusi = 178 microseconds
```

3.4. Dimensi > 3, n = 64

```
Masukkan banyak titik
64
Masukkan banyak dimensi
4
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 134.967
Titik 1 = {-270, 0, 60, 128}
Titik 2 = {-360, 60, 120, 182}
Banyaknya operasi euclidean = 2017
Waktu eksekusi = 865 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 134.967
Titik 1 = {-360, 60, 120, 182}
Titik 2 = {-270, 0, 60, 128}
Banyaknya operasi euclidean = 1082
Waktu eksekusi = 397 microseconds
```

```
Masukkan banyak titik
64
Masukkan banyak dimensi
5
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 267.045
Titik 1 = {-520, 71, 462, -108, -240}
Titik 2 = {-560, 0, 350, -120, -468}
Banyaknya operasi euclidean = 2017
Waktu eksekusi = 833 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 267.045
Titik 1 = {-560, 0, 350, -120, -468}
Titik 2 = {-520, 71, 462, -108, -240}
Banyaknya operasi euclidean = 1603
Waktu eksekusi = 647 microseconds
```

3.5. Dimensi = 3, n =128

```
Masukkan banyak titik
128
Masukkan banyak dimensi
3
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 46.8188
Titik 1 = {480, 80, 336}
Titik 2 = {440, 84, 360}
Banyaknya operasi euclidean = 8129
Waktu eksekusi = 2467 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 46.8188
Titik 1 = {440, 84, 360}
Titik 2 = {480, 80, 336}
Banyaknya operasi euclidean = 1482
Waktu eksekusi = 508 microseconds
```

```
Masukkan banyak titik
128
Masukkan banyak dimensi
3
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 14.1421
Titik 1 = {0, -105, 450}
Titik 2 = {0, -95, 440}
Banyaknya operasi euclidean = 8129
Waktu eksekusi = 2504 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 14.1421
Titik 1 = {0, -95, 440}
Titik 2 = {0, -105, 450}
Banyaknya operasi euclidean = 1310
Waktu eksekusi = 490 microseconds
```

3.6. Dimensi bukan 3, n = 128

```
Masukkan banyak titik
128
Masukkan banyak dimensi
2
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 4
Titik 1 = {-234, 0}
Titik 2 = {-238, 0}
Banyaknya operasi euclidean = 8129
Waktu eksekusi = 2614 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 4
Titik 1 = {-238, 0}
Titik 2 = {-234, 0}
Banyaknya operasi euclidean = 556
Waktu eksekusi = 216 microseconds
```

```
Masukkan banyak titik
128
Masukkan banyak dimensi
6
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 162.678
Titik 1 = {240, -600, -513, -171, 114, -90}
Titik 2 = {315, -600, -456, -228, 204, -11}
Banyaknya operasi euclidean = 8129
Waktu eksekusi = 3532 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 162.678
Titik 1 = {240, -600, -513, -171, 114, -90}
Titik 2 = {315, -600, -456, -228, 204, -11}
Banyaknya operasi euclidean = 5771
Waktu eksekusi = 2694 microseconds
```

3.7. Dimensi = 3, n =1000

```
Masukkan banyak titik
1000
Masukkan banyak dimensi
3
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 2
Titik 1 = {-108, 0, 544}
Titik 2 = {-110, 0, 544}
Banyaknya operasi euclidean = 499501
Waktu eksekusi = 149044 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 2
Titik 1 = {-110, 0, 544}
Titik 2 = {-108, 0, 544}
Banyaknya operasi euclidean = 43412
Waktu eksekusi = 11880 microseconds
```

```
Masukkan banyak titik
1000
Masukkan banyak dimensi
3
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 5.09902
Titik 1 = {-260, -196, 252}
Titik 2 = {-264, -195, 255}
Banyaknya operasi euclidean = 499501
Waktu eksekusi = 153112 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 5.09902
Titik 1 = {-264, -195, 255}
Titik 2 = {-260, -196, 252}
Banyaknya operasi euclidean = 38314
Waktu eksekusi = 10605 microseconds
```

3.8. Dimensi bukan 3, n = 1000

```
Masukkan banyak titik
1000
Masukkan banyak dimensi
5
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 36.5103
Titik 1 = {342, -200, 486, -26, 0}
Titik 2 = {324, -197, 504, 0, 0}
Banyaknya operasi euclidean = 499501
Waktu eksekusi = 182435 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 36.5103
Titik 1 = {324, -197, 504, 0, 0}
Titik 2 = {342, -200, 486, -26, 0}
Banyaknya operasi euclidean = 139375
Waktu eksekusi = 45728 microseconds
```

```
Masukkan banyak titik
1000
Masukkan banyak dimensi
8
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000

Hasil menggunakan algoritma brute force:
Jarak terdekat = 217.143
Titik 1 = {-588, 256, -624, -308, 132, 119, -600, 448}
Titik 2 = {-540, 408, -585, -272, 171, 50, -530, 360}
Banyaknya operasi euclidean = 499501
Waktu eksekusi = 227896 microseconds

Hasil menggunakan algoritma divide and conquer:
Jarak terdekat = 217.143
Titik 1 = {-588, 256, -624, -308, 132, 119, -600, 448}
Titik 2 = {-540, 408, -585, -272, 171, 50, -530, 360}
Banyaknya operasi euclidean = 400071
Waktu eksekusi = 169656 microseconds
```

3.9. Input tidak valid

```
Masukkan banyak titik
1
Masukan salah, silahkan masukkan kembali
3
Masukkan banyak dimensi
1
Masukan salah, silahkan masukkan kembali
-3
Masukan salah, silahkan masukkan kembali
3
Masukkan batas ujung (batas ujung diasumsikan simetrik)
1000
```

4. Referensi

1. Closest Pair of Points using Divide and Conquer algorithm - GeeksforGeeks
2. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf
3. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf

5. Lampiran

Link repository: https://github.com/MuhLibri/Tucil2_13521047.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	✓	