

LAPORAN TUGAS KECIL 2 IF2211

# **Mencari Pasangan Titik Terdekat 3D dengan *Algoritma Divide and Conquer***



Ditujukan untuk memenuhi salah satu tugas kecil mata kuliah IF2211 Strategi  
Algoritma pada Semester II Tahun Akademik 2022/2023

Disusun oleh:

**Edia Zaki Naufal Ilman**

**13521141**

**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2023**

# BAB I

## ALGORITMA

### 1.1. Algoritma *Brute Force*

Algoritma *brute force* adalah suatu pendekatan algoritma yang lempang (*straightforward*) untuk memecahkan suatu persoalan. Algoritma ini memecahkan suatu persoalan dengan cara yang sangat sederhana, langsung (*direct*), dan jelas (*obvious*). Pendekatan *brute force* membutuhkan volume komputasi dan waktu penyelesaian yang relative cukup besar. Meskipun begitu, algoritma *brute force* dapat menyelesaikan hamper semua persoalan dan sangat cocok untuk persoalan-persoalan kecil.

### 1.2. Algoritma *Divide and Conquer*

Algoritma *divide and conquer* adalah suatu pendekatan algoritma untuk memecahkan suatu persoalan dengan cara yang membagi persoalan tersebut (*divide*) sehingga menjadi suatu bagian yang lebih kecil untuk dapat diselesaikan (*conquer*). Pendekatan *divide and conquer* ini memiliki 3 fase dalam implementasinya, yaitu fase *divide* atau membagi-bagi persoalan menjadi lebih kecil, fase *conquer* atau menyelesaikan persoalan yang sudah dibagi, dan terakhir fase *combine* atau menggabungkan solusi sehingga membentuk solusi persoalan semula. Algoritma ini lebih efisien dari algoritma *brute force*.

Berikut merupakan Langkah-langkah algoritma pada program ini:

1. Urutkan kumpulan titik koordinat dari kecil hingga besar menggunakan algoritma *quicksort*
2. Membagi kumpulan titik menjadi dua bagian yang sama dan menentukan garis tengah yang membaginya
3. Mencari jarak terpendek pada masing-masing bagian
4. Mencari jarak terpendek antar titik kedua bagian yang dekat dengan garis pembagi
5. Bandingkan semua jarak terdekat yang didapatkan

### 1.3. *Quicksort*

*Quicksort* merupakan salah satu algoritma pengurutan yang menggunakan pendekatan *divide and conquer*. Algoritma ini termasuk kedalam category *hard split/easy join* yang berarti pembagian persoalan relatif lebih susah dan penggabungan relatif lebih mudah. *Quicksort* ini diimplementasikan dengan cara membagi dua buah larik yang dipisahkan oleh sebuah *pivot* atau acuan dimana larik pertama terdiri dari bagian yang lebih kecil dari acuan, dan larik kedua terdiri dari bagian yang lebih besar dari acuan. Setelah dibagi-bagi hingga bagian yang lebih kecil, setiap

potongan larik kemudian akan diurutkan dan digabungkan kembali.

Berikut merupakan langkah-langkah algoritma *quicksort* pada program ini:

1. Jadikan larik urutan pertama sebagai pivot atau acuan
2. Tukar elemen-elemen yang lebih kecil dari pivot dengan elemen-elemen yang lebih besar dari pivot
3. Bagi larik menjadi dua bagian dengan pivot sebagai pembagi
4. Lakukan iterasi dari Langkah pertama hingga ukuran larik menjadi satu atau dua
5. Urutkan larik yang sudah kecil
6. Gabungkan semua hasil pengurutan

## BAB II

### SOURCE

### CODE

#### 2.1 Point Class Header File (point.hpp)

```
#ifndef __POINT__HPP__
#define __POINT__HPP__

class Point
{
private:
    int x;
    int y;
    int z;
public:
    Point();
    Point(int, int, int);
    ~Point();

    void operator=(const Point&);

    int getX();
    int getY();
    int getZ();
    void setX(int);
    void setY(int);
    void setZ(int);

    double getDistance(Point);
    void printPoint();
    //void quicksort(array<Point, 1000>);
};

#endif
```

## 2.2. Point Class File (point.cpp)

```
#include "point.hpp"
#include <cstdlib>
#include <iostream>
#include <cmath>
using namespace std;

Point::Point() {
    x = 0;
    y = 0;
    z = 0;
}

Point::Point(int x, int y, int z) {
    this->x = x;
    this->y = y;
    this->z = z;
}

Point::~Point() {}

int Point::getX() {
    return x;
}

int Point::getY() {
    return y;
}

int Point::getZ() {
    return z;
}

void Point::setX(int x) {
    this->x = x;
}

void Point::setY(int y) {
    this->y = y;
}

void Point::setZ(int z) {
    this->z = z;
}

void Point::operator=(const Point& other) {
    this->x = other.x;
    this->y = other.y;
```

```

        this->z = other.z;
    }

double Point::getDistance(Point other){
    int dX = pow(abs(this->x - other.getX()), 2);
    int dY = pow(abs(this->y - other.getY()), 2);
    int dZ = pow(abs(this->z - other.getZ()), 2);
    double d = sqrt(dX + dY + dZ);

    return d;
}

void Point::printPoint(){
    cout << "Coordinates - X: " << x << " | Y: " << y << " | Z: " << z << endl;
}

```

## 2.3. Utilities Header File(utilities.hpp)

```

#ifndef __UTILITIES_HPP__
#define __UTILITIES_HPP__

#include "point.cpp"

#define MAX_ARR 1000
#define MIN_COOR -500
#define MAX__COOR 500

void quicksort(Point arr[], int n);

void BruteForce(Point arr[], int n);

void DivideAndConquer(Point arr[], int n);

void split(Point parent[], Point child1[], Point child2[], int n);

#endif

```

## 2.4. Utilities File(utilities.cpp)

```

#include "utilities.hpp"

void quicksort(Point arr[], int n){

```

```

Point pivot = arr[0];
//cout << "Begin Quicksort " << endl << endl;
//pivot.printPoint();
if(n == 1){

}

else if(n == 2){
    if(arr[0].getX() > arr[1].getX()){
        arr[0] = arr[1];
        arr[1] = pivot;
    }

}

else{
    int p = 1;
    int q = n - 1;
    bool flagBigger = false;
    bool flagSmaller = false;
    int count1 = 0;
    int count2 = 0;

    //Run positioning for splitting
    while(p < q){
        for(; p < n; p++){
            if(arr[p].getX() > pivot.getX()){
                //arr[p].printPoint();
                //cout << "Is bigger than pivot" << endl;
                flagBigger = true;
                count1++;
                break;
            }else{
                //arr[p].printPoint();
                //cout << "Is not bigger than pivot" << endl;
            }
        }

        // Case if pivot is the biggest
        if(!flagBigger){
            p = 0;
        }

        for(; q >= 1; q--){
            if(arr[q].getX() <= pivot.getX()){
                //arr[q].printPoint();
            }
        }
    }
}

```

```

        //cout << "Is smaller than pivot" << endl;
        flagSmaller = true;
        count2++;
        break;
    }
}

// Break out loop if q and p already cross
if(q <= p){
    break;
}

// Case if pivot is the smallest
if(!flagSmaller){
    q = 0;
}

//Swap q and p (smaller and bigger)
if(flagBigger && flagSmaller){
    swap(arr[p], arr[q]);

    p++;
    q--;
}else{ //if pivot is the biggest or smallest
    break;
}

}

//Swap the pivot if it's not the smallest
if(flagSmaller){
    arr[0] = arr[q];
    arr[q] = pivot;
}

Point firstHalf[MAX_ARR];
Point secondHalf[MAX_ARR];

//Create the first half of points (if pivot is not smallest)

```



```

int i = 0;
if(flagSmaller){
    //cout << "FIRST HALF" << endl;
    for(i = 0; i < q; i++){
        firstHalf[i] = arr[i];
        // firstHalf[i].printPoint();
    }

}

i++;

//Create the second half of points (if pivot is not biggest)
if(flagBigger){
    //cout << "SECOND HALF" << endl;
    for(int j = 0; i < n; j++){
        secondHalf[j] = arr[i];
        // secondHalf[j].printPoint();
        i++;
    }
}

//Recursive for bot halves
if(flagSmaller){
    quicksort(firstHalf, q);
}
if(flagBigger){
    quicksort(secondHalf, n - (q + 1));
}

//Merging all the ordered halves
Point result[MAX_ARR];
int j = 0;
if(flagSmaller){
    //cout << "First merge\n";
    for(j = 0; j < q; j++){
        arr[j] = firstHalf[j];
        //arr[j].printPoint();
    }
}

arr[j] = pivot;
j++;

```

```

        if(flagBigger){
            //cout << "second merge\n";
            for(int k = 0; j < n; k++){
                arr[j] = secondHalf[k];
                //arr[j].printPoint();
                j++;
            }
        }
    }
}

void BruteForce(Point arr[], int n){
    //Brute Force Way
    double MIN = arr[0].getDistance(arr[1]);
    Point T1 = arr[0];
    Point T2 = arr[1];
    int calculation = 0;

    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if(i != j && arr[i].getDistance(arr[j]) < MIN){
                MIN = arr[i].getDistance(arr[j]);
                calculation++;
                T1 = arr[i];
                T2 = arr[j];
            }
            calculation++;
        }
    }

    cout << "BRUTE FORCE: " << MIN << endl;
    cout << "Jumlah perhitungan Euclidean: " << calculation << endl;
    cout << "Titik 1: (" << T1.getX() << ", " << T1.getY() << ", " << T1.getZ() <<
    ")" << endl;
    cout << "Titik 2: (" << T2.getX() << ", " << T2.getY() << ", " << T2.getZ() <<
    ")" << endl;

}

void DivideAndConquer(Point arr[], int n){
    Point leftSide[MAX_ARR / 2];
    Point rightSide[MAX_ARR / 2];

```

```

Point T1 = arr[0];
Point T2 = arr[1];
int leftMax = round(n / 2) - 1;
int count = 0;
int calculation = 0;

//Split array of points
split(arr, leftSide, rightSide, n);
double strip = (rightSide[0].getX() - leftSide[leftMax].getX()) +
leftSide[leftMax].getX();

//Shortest Distance on Left
double LeftMin = -1;
Point LeftT1 = Point();
Point LeftT2 = Point();
if(round(n / 1) > 1){
    LeftT1 = leftSide[0];
    LeftT2 = leftSide[1];
    LeftMin = leftSide[0].getDistance(leftSide[1]);
    calculation++;
    for(int i = 0; i < round(n / 2); i++){
        for(int j = 0; j < round(n / 2); j++){
            if(i < j && leftSide[i].getDistance(leftSide[j]) < LeftMin){
                LeftMin = leftSide[i].getDistance(leftSide[j]);
                calculation++;
                LeftT1 = leftSide[i];
                LeftT2 = leftSide[j];
            }
            calculation++;
        }
    }
}

//Shortest Distance on Right
double RightMin = rightSide[0].getDistance(rightSide[1]);
Point RightT1 = rightSide[0];
Point RightT2 = rightSide[1];
calculation++;
for(int i = 0; i < n - round(n / 2); i++){
    for(int j = 0; j < n - round(n / 2); j++){
        if(i < j && rightSide[i].getDistance(rightSide[j]) < RightMin){
            RightMin = rightSide[i].getDistance(rightSide[j]);
            calculation++;
        }
    }
}

```

```

        RightT1 = rightSide[i];
        RightT2 = rightSide[j];
    }
    calculation++;
}
}

//Shortest distance between left and right
double universalMin;
if(LeftMin == -1 || RightMin <= LeftMin){
    universalMin = RightMin;
    T1 = RightT1;
    T2 = RightT2;
}else{
    universalMin = LeftMin;
    T1 = LeftT1;
    T2 = LeftT2;
}
//Shortest distance near strip
count = 0;
for(int i = 0; i < round(n / 2); i++){
    if(leftSide[i].getX() >= strip - universalMin && leftSide[i].getX() <= strip
+ universalMin){
        for(int j = 0; j < n - round(n / 2); j++){
            if(rightSide[i].getX() >= strip - universalMin && rightSide[i].getX()
<= strip + universalMin && leftSide[i].getDistance(rightSide[j]) < universalMin){
                universalMin = leftSide[i].getDistance(rightSide[j]);
                calculation++;
                T1 = leftSide[i];
                T2 = rightSide[j];
            }
            calculation++;
        }
    }
}

cout << "DIVIDE & CONQUER: " << universalMin << endl;
cout << "Jumlah perhitungan Euclidean: " << calculation << endl;
cout << "Titik 1: (" << T1.getX() << ", " << T1.getY() << ", " << T1.getZ() <<
")" << endl;
cout << "Titik 2: (" << T2.getX() << ", " << T2.getY() << ", " << T2.getZ() <<
")" << endl;
}

```

```

void split(Point parent[], Point child1[], Point child2[], int n){
    int count = 0;
    // Left Points
    for(int i = 0; i < round(n/2); i++){
        child1[i] = parent[count];
        //leftSide[i].printPoint();
        count++;
    }
    //Right points
    for(int i = 0; i < n - round(n / 2); i++){
        child2[i] = parent[count];
        //rightSide[i].printPoint();
        count++;
    }
}

```

## 2.5. Main Driver File(main.cpp)

```

#include <iostream>
#include <time.h>
#include <ctime>
#include <chrono>
#include "utilities.cpp"

using namespace std;
using chrono::duration_cast;
using chrono::duration;
using chrono::milliseconds;

typedef chrono::high_resolution_clock Clock;

int main(){
    //Point p[MAX_ARR];
    int n;
    Point points[MAX_ARR];
    Point* pointsPtr = points;
    Point pointsByX[MAX_ARR];
    int Rx, Ry, Rz;
    srand(time(NULL));

    //Opening

```

```

cout << "Selamat datang!\n" << endl;
cout << "Keterangan:" << endl;
cout << "1) Range Koordinat: (-500) - 500\n2) Maximum titik: 1000\n" << endl;
cout << "Masukan jumlah titik: ";
cin >> n;
cout << endl;

//Generate random points
for(int i = 0; i < n; i++){
    Rx = rand() % (MAX__COORD - MIN_COORD + 1) + MIN_COORD;
    Ry = rand() % (MAX__COORD - MIN_COORD + 1) + MIN_COORD;
    Rz = rand() % (MAX__COORD - MIN_COORD + 1) + MIN_COORD;

    points[i] = Point(Rx, Ry, Rz);
    //points[i].printPoint();

}

//Sort the array
quicksort(points, n);
// cout<<"RESULTS\n";
// for(int i = 0; i < n; i++){
//     pointsByX[i].printPoint();
// }

//Brute Force Way
auto start_s = Clock::now();
BruteForce(points, n);
auto stop_s = Clock::now();
duration<double, milli> exec = stop_s - start_s;
cout << "Execution time: " << exec.count() << " ms" << endl << endl;

//Divide and Conquer Way
start_s = Clock::now();
DivideAndConquer(points, n);
stop_s = Clock::now();
exec = stop_s - start_s;
cout << "Execution time: " << exec.count() << " ms" << endl << endl;

return 0;
}

```



## BAB III

### EKSPERIMEN

#### 3.1. Hasil Input dan Output Program Saat $n = 16$

```
Selamat datang!

Keterangan:
1) Range Koordinat: (-500) - 500
2) Maximum titik: 1000

Masukan jumlah titik: 16

BRUTE FORCE: 92.2497
Jumlah perhitungan Euclidean: 260
Titik 1: (-361, -13, 211)
Titik 2: (-347, 20, 126)
Execution time: 2.287 ms

DIVIDE & CONQUER: 92.2497
Jumlah perhitungan Euclidean: 151
Titik 1: (-361, -13, 211)
Titik 2: (-347, 20, 126)
Execution time: 1.002 ms
```

#### 3.2. Hasil Input dan Output Program Saat $n = 64$

```
Selamat datang!

Keterangan:
1) Range Koordinat: (-500) - 500
2) Maximum titik: 1000

Masukan jumlah titik: 64

BRUTE FORCE: 21.0713
Jumlah perhitungan Euclidean: 4104
Titik 1: (156, -496, -34)
Titik 2: (167, -496, -52)
Execution time: 1.985 ms

DIVIDE & CONQUER: 21.0713
Jumlah perhitungan Euclidean: 2223
Titik 1: (156, -496, -34)
Titik 2: (167, -496, -52)
Execution time: 2.427 ms
```

#### 3.3. Hasil Input dan Output Program Saat $n = 128$



```
Selamat datang!

Keterangan:
1) Range Koordinat: (-500) - 500
2) Maximum titik: 1000

Masukan jumlah titik: 128

BRUTE FORCE: 31.8277
Jumlah perhitungan Euclidean: 16398
Titik 1: (438, -288, -272)
Titik 2: (452, -271, -295)
Execution time: 8.1 ms

DIVIDE & CONQUER: 31.8277
Jumlah perhitungan Euclidean: 8593
Titik 1: (438, -288, -272)
Titik 2: (452, -271, -295)
Execution time: 3.008 ms
```

### 3.4. Hasil Input dan Output Program Saat n = 1000

```
Selamat datang!

Keterangan:
1) Range Koordinat: (-500) - 500
2) Maximum titik: 1000

Masukan jumlah titik: 1000

BRUTE FORCE: 7.14143
Jumlah perhitungan Euclidean: 1000011
Titik 1: (390, 170, 1)
Titik 2: (391, 169, 8)
Execution time: 324.895 ms

DIVIDE & CONQUER: 7.14143
Jumlah perhitungan Euclidean: 502519
Titik 1: (390, 170, 1)
Titik 2: (391, 169, 8)
Execution time: 123.097 ms
```

## **REFERENSI**

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf)

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf)

## LAMPIRAN

Link github:

[https://github.com/Ezaaan/Tucil2\\_13521141](https://github.com/Ezaaan/Tucil2_13521141)

Tabel pengerjaan:

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