LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA
PENCARIAN PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA DIVIDE AND CONQUER

Disusun oleh:

Fazel Ginanda (13521098)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

Daftar Isi

BAB I
PENDAHULUAN

Setiap titik P di dalam ruang dinyatakan dengan koordinat P = (x,y,z). Dua buah titik dapat ditentukan jaraknya menggunakan rumus Euclidean. Rumus Euclidean untuk menghitung jarak di dalam ruang dinyatakan sebagai berikut.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Pada tugas kecil ini, penulis membuat program menggunakan bahasa C untuk mencari pasangan titik dalam dimensi tiga yang memiliki jarak terdekat menggunakan algoritma *divide and conquer*. Titik-titik yang menjadi masukan program dibangkitkan secara acak menggunakan *library* yang sudah tersedia. Pengguna program hanya memberikan masukan banyak titik yang dikomputasikan. Keluaran dari program ini berupa pasangan titik yang memiliki jarak terdekat beserta jaraknya, banyaknya operasi perhitungan Euclidean yang dilakukan, serta waktu riil eksekusi program. Selain itu juga digunakan algoritma *brute force* untuk membandingkan hasil eksekusi kedua algoritma.

BAB II
ALGORITMA

Algoritma *divide and conquer* adalah algoritma yang membagi persoalan menjadi beberapa upa-persoalan menjadi beberapa upa-persoalan yang mirip dengan persoalan semula tetapi berukuran lebih kecil. Kumpulan upa-persoalan tersebut diselesaikan satu per satu dan digabungkan kembali sehingga membentuk solusi persoalan semula. Setiap upa-persoalan memiliki karakteristik yang sama sehingga tepat untuk dinyatakan secara rekursif.

Algoritma yang digunakan oleh penulis dalam merancang program dalam tugas kecil ini dijelaskan sebagai berikut.

1. Terima masukan berupa banyak titik yang akan dibangkitkan oleh pembangkit bilangan acak pada *library.*
2. Buat larik statis dengan kapasitas 1000 elemen. Elemen dari setiap senarai tersebut adalah titik dalam dimensi tiga.
3. Urutkan larik menaik berdasarkan nilai absis menggunakan algoritma *selection sort.*
4. Jika elemen larik hanya dua, maka hitung jarak antara kedua titik tersebut menggunakan rumus Euclidean. Dua titik tersebut juga merupakan solusi pasangan titik terdekat.
5. Jika elemen larik terdapat sebanyak tiga, maka hitung jarak antara setiap titik dengan dua titik lainnya. Kemudian bandingkan semua jarak tersebut untuk memperoleh jarak terkecil.
6. Jika elemen larik terdapat lebih dari tiga, maka bagi dua kumpulan titik tersebut menjadi dua himpunan, yaitu S1 dan S2.
7. Cari pasangan titik terdekat beserta jaraknya dalam setiap himpunan secara rekursif.
8. Bandingkan jarak terdekat dari pasangan titik hasil perhitungan pada S1 dengan S2. Pasangan titik yang memiliki jarak terdekat akan disimpan dalam solusi sementara.
9. Cari absis dari garis tengah yang terletak di antara kedua himpunan.
10. Tentukan titik-titik yang berada di S1 atau S2 yang berjarak paling jauh sebesar d dari garis tengah.
11. Masukkan titik-titik tersebut ke dalam suatu himpunan yang disebut Sstrip.
12. Jika selisih absis atau selisih ordinat dari pasangan titik dalam himpunan Sstrip tersebut kurang dari jarak terdekat yang diperoleh sebelumnya, maka hitung jarak pasangan titik

tersebut. Jika pasangan titik tersebut memiliki jarak yang lebih dekat, maka ubah solusi sebelumnya.

BAB III
IMPLEMENTASI KODE PROGRAM

3.1 main.c

```c
#include "Point.h"
#include "ListPoint.h"
#include <stdio.h>
#include <time.h>

int main()
{
    int n;
    printf("PROGRAM PENCARIAN PASANGAN TITIK TERDEKAT\n");
    printf("Masukkan banyak titik (n): ");
    do
    {
        scanf("%d", &n);
        if (n < 2)
        {
            printf("Masukan salah. Titik minimal sebanyak 2. Ulangi!\n");
            printf("Masukkan banyak titik (n): ");
        }
    } while (n < 2);

    ListPoint l;
    CreateListPointRandom(&l, n);
```

```c
    /* MENCARI DENGAN ALGORITMA BRUTE FORCE */
    Point p1BF, p2BF;
    float minBF;
    int countBF;
    clock_t startBF, endBF;
    double timeBF;

    startBF = clock();
    FindClosestPairBF(&l, &p1BF, &p2BF, &minBF, &countBF);
    endBF = clock();
    timeBF = ((double)(endBF - startBF)) / CLOCKS_PER_SEC;

    printf("\n");
    printf("Hasil pencarian pasangan titik terdekat berdasarkan algoritma brute force:\n");
    printf("1. Pasangan titik\n");
    printf("    ");
    printf("Titik pertama: ");
    PrintPoint(p1BF);
    printf("\n");
    printf("    ");
    printf("Titik kedua: ");
    PrintPoint(p2BF);
    printf("\n");
    printf("    ");
    printf("Jarak pasangan titik terdekat: %f\n", minBF);
    printf("2. Banyak perhitungan rumus Euclidean: %d\n", countBF);
    printf("3. Waktu riil: %f\n", timeBF);
```

```c
    /* MENCARI DENGAN ALGORITMA DIVIDE AND CONQUER */
    Point p1DC, p2DC;
    float minDC;
    int countDC;
    clock_t startDC, endDC;
    float timeDC;

    startDC = clock();
    selectionSortX(&l, 0, n - 1);
    FindClosestPairDC(&l, n, &p1DC, &p2DC, &minDC, &countDC);
    endDC = clock();
    timeDC = ((double)(endDC - startDC)) / CLOCKS_PER_SEC;

    printf("\n");
    printf("Hasil pencarian pasangan titik terdekat berdasarkan algoritma divide and conquer:\n");
    printf("1. Pasangan titik terdekat dan jaraknya\n");
    printf("   ");
    printf("Titik pertama: ");
    PrintPoint(p1DC);
    printf("\n");
    printf("   ");
    printf("Titik kedua: ");
    PrintPoint(p2DC);
    printf("\n");
    printf("   ");
    printf("Jarak pasangan titik terdekat: %f\n", minDC);
    printf("2. Banyak perhitungan rumus Euclidean: %d\n", countDC);
    printf("3. Waktu riil: %f\n", timeDC);
}
```

3.2 Point.h

```c
/* File: Point.h */
/* *** Definisi ABSTRACT DATA TYPE Point *** */

#ifndef POINT_H
#define POINT_H

#include "boolean.h"

typedef struct
{
    float X; /* absis   */
    float Y; /* ordinat */
    float Z; /* aplikat */
} Point;

/* *** Notasi Akses: Selektor Point *** */
#define Absis(P) (P).X
#define Ordinat(P) (P).Y
#define Aplikat(P) (P).Z

/* *** DEFINISI PROTOTIPE PRIMITIF *** */
/* *** Konstruktor membentuk Point *** */
Point MakePoint(float X, float Y, float Z);
/* Membentuk sebuah Point dari komponen-komponennya */
/* *** Konstruktor membentuk Point dengan komponen yang dibangkitkan secara acak *** */
Point MakeRandomPoint(void);

```

```c
/* *** KELOMPOK Interaksi dengan I/O device, BACA/TULIS  *** */
void PrintPoint(Point P);
/* Nilai P ditulis ke layar dengan format "(X,Y)"
   tanpa spasi, enter, atau karakter lain di depan, belakang,
   atau di antaranya
   Output X dan Y harus dituliskan dalam bilangan riil dengan 2 angka di belakang koma.
*/
/* I.S. P terdefinisi */
/* F.S. P tertulis di layar dengan format "(X,Y)" */

/* *** KELOMPOK OPERASI LAIN TERHADAP TYPE *** */
float Distance(Point P1, Point P2);
/* Menghitung jarak antara dua titik dengan rumus Euclidean */

/* *** Kelompok operasi relasional terhadap POINT *** */
boolean Equal(Point P1, Point P2);
/* Mengirimkan true jika P1 = P2 : absis, ordinat, aplikatnya sama */

/* *** Operasi tambahan *** */
float ABS(float a, float b);
/* Mengembalikan nilai mutlak dari selisih dua bilangan real */

#endif
```

3.3 Point.c

```c
/* File: Point.c */
/* Implementasi dari Point.h */

#include "Point.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Implementasi konstruktor membentuk Point */
Point MakePoint(float X, float Y, float Z)
{
    /* KAMUS */
    Point p;

    /* ALGORITMA */
    Absis(p) = X;
    Ordinat(p) = Y;
    Aplikat(p) = Z;
    return p;
}

Point MakeRandomPoint(void)
{
    /* KAMUS */
    Point p;

    /* ALGORITMA */
    Absis(p) = rand();
    Ordinat(p) = rand();
    Aplikat(p) = rand();
    return p;
}

```

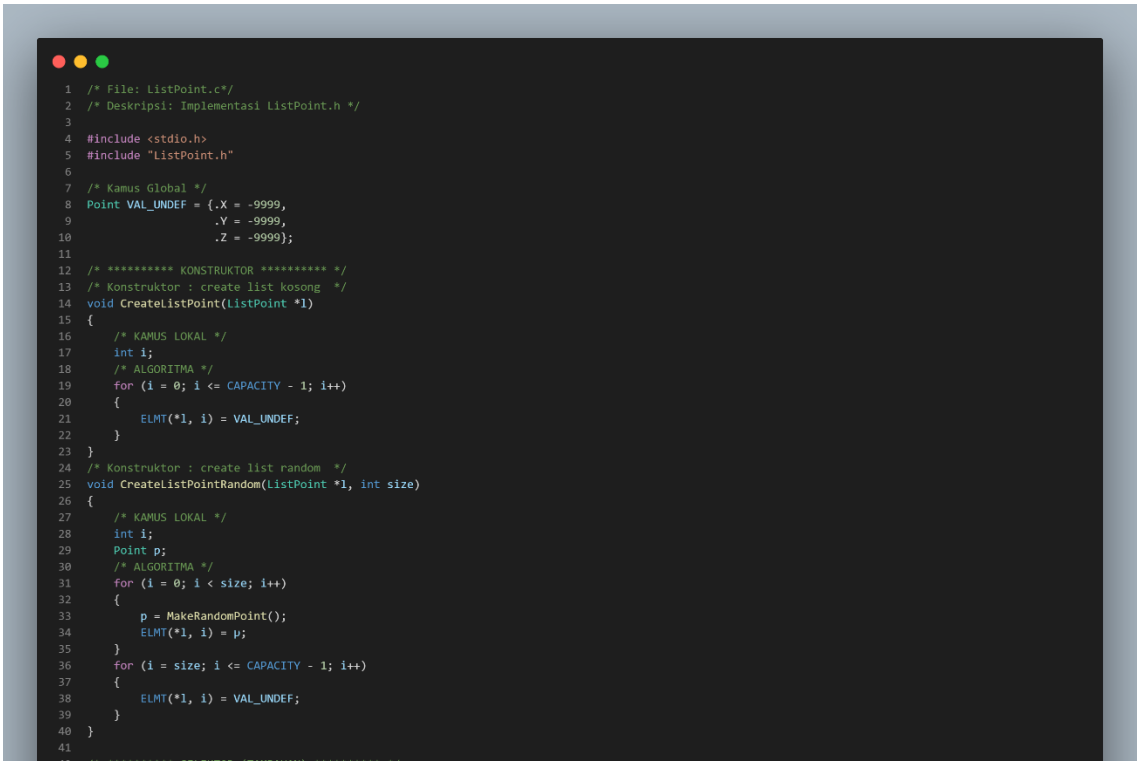
```c
/* *** KELOMPOK Interaksi dengan I/O device, BACA/TULIS  *** */
void PrintPoint(Point P)
{
    /*KAMUS*/
    /* ALGORITMA */
    printf("(%.3f,%.3f,%.3f)", Absis(P), Ordinat(P), Aplikat(P));
}

/* *** KELOMPOK OPERASI LAIN TERHADAP TYPE *** */
float Distance(Point P1, Point P2)
{
    /*KAMUS*/
    /*ALGORITMA*/
    return (sqrtf(powf(Absis(P2) - Absis(P1), 2) + powf(Ordinat(P2) - Ordinat(P1), 2) + powf(Aplikat(P2) - Aplikat(P1), 2)));
}

/* *** Kelompok operasi relasional terhadap POINT *** */
boolean Equal(Point P1, Point P2)
{
    /*KAMUS*/
    /* ALGORITMA */
    return ((Absis(P1) == Absis(P2)) && (Ordinat(P1) == Ordinat(P2)) && (Aplikat(P1) == Aplikat(P2)));
}

/* Operasi tambahan */
float ABS(float a, float b)
{
    if (a > b)
    {
        return a - b;
    }
    else
    {
        return o - a;
    }
}
```

3.4 ListPoint.h

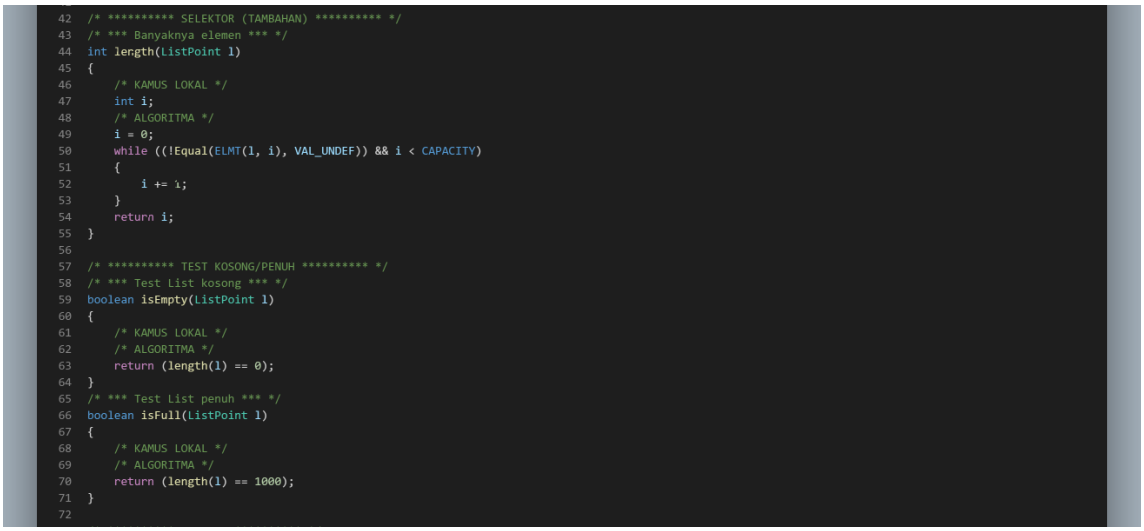
```c
/* File: ListPoint.h */
/* Definisi ADT ListPoint */
/* Penempatan elemen selalu rapat kiri */
/* Banyaknya elemen didefinisikan secara implisit, memori array statik */

#ifndef LISTPOINT_H
#define LISTPOINT_H

#include "boolean.h"
#include "Point.h"

/*  Kamus Umum */
#define CAPACITY 1000
/* Kapasitas penyimpanan */
#define IDX_UNDEF -1
/* Indeks tak terdefinisi*/
extern Point VAL_UNDEF;
/* Deklarasi nilai elemen tak terdefinisi*/

/* Definisi elemen dan koleksi objek */
typedef Point ElType; /* type elemen List */
typedef struct
{
   ElType contents[CAPACITY]; /* memori tempat penyimpan elemen (container) */
} ListPoint;
/* Indeks yang digunakan [0..CAPACITY-1] */
/* Jika l adalah ListPoint, cara deklarasi dan akses: */
/* Deklarasi : l : ListPoint */
/* Maka cara akses:
   ELMT(l,i) untuk mengakses elemen ke-i */
/* Definisi :
   List kosong: semua elemen bernilai VAL_UNDEF
   Definisi elemen pertama: ELMT(l,i) dengan i=0 */

/* ********** SELEKTOR ********** */
#define ELMT(l, i) (l).contents[(i)]

```

```c
/* ********** KONSTRUKTOR ********** */
/* Konstruktor : create List kosong  */
void CreateListPoint(ListPoint *l);
/* I.S. l sembarang */
/* F.S. Terbentuk List l kosong dengan kapasitas CAPACITY */
/* Proses: Inisialisasi semua elemen List l dengan VAL_UNDEF */
/* *** Konstruktor List Random *** */
void CreateListPointRandom(ListPoint *l, int size);
/* I.S. l sembarang, capacity > 0 */
/* F.S. Terbentuk list dinamis l dengan ukuran sebesar size */
/* Setiap elemen list diinisialisasi secara acak */

/* ********** SELEKTOR (TAMBAHAN) ********** */
/* *** Banyaknya elemen *** */
int length(ListPoint l);
/* Mengirimkan banyaknya elemen efektif List */
/* Mengirimkan nol jika List kosong */

/* ********** TEST KOSONG/PENUH ********** */
/* *** Test List kosong *** */
boolean isEmpty(ListPoint l);
/* Mengirimkan true jika List l kosong, mengirimkan false jika tidak */
/* *** Test List penuh *** */
boolean isFull(ListPoint l);
/* Mengirimkan true jika List l penuh, mengirimkan false jika tidak */

/* ********** SORTING ********** */
void selectionSortX(ListPoint *l, int i, int j);
/* I.S. l boleh kosong */
/* F.S. l terurut membesar berdasarkan absis */
/* Proses : Mengurutkan l dengan salah satu algoritma selection sort berdasarkan absis */
void partisiX(ListPoint *l, int i, int j);
/* Mempartisi larik l[i..j] dengan cara mencari elemen minimum di dalam l[i..j] */
/* Menempatkan elemen terkecil sebagai elemen pertama larik */
/* Masukan: l[i..j] sudah terdefinisi elemen-elemennya */
/* Luaran: l[i..j] dengan l[i] adalah elemen minimum. */
void selectionSortY(ListPoint *l, int i, int j);
/* I.S. l boleh kosong */
/* F.S. l terurut membesar berdasarkan ordinat */
/* Proses : Mengurutkan l dengan salah satu algoritma selection sort berdasarkan ordinat */
void partisiY(ListPoint *l, int i, int j);
/* Mempartisi larik A[i..j] dengan cara mencari elemen minimum di dalam l[i..j] */
/* Menempatkan elemen terkecil sebagai elemen pertama larik */
/* Masukan: l[i..j] sudah terdefinisi elemen-elemennya */
/* Luaran: l[i..j] dengan l[i] adalah elemen minimum. */

/* ********** OPERASI LAIN ********** */
void copyList(ListPoint lIn, int firstIdxCopy, int lastIdxCopy, ListPoint *lOut);
/* I.S. lIn terdefinisi tidak kosong, lOut sembarang */
/* F.S. lOut berisi salinan dari lIn
        Elemen yang disalin mulai dari indeks ke-firstIdx sampai lastIdx */
/* Proses : Menyalin isi lIn ke lOut */

/* ********** MENAMBAH ELEMEN DI AKHIR ********** */
/* *** Menambahkan elemen terakhir *** */
void insertLast(ListPoint *l, ElType val);
/* Proses: Menambahkan val sebagai elemen terakhir List */
/* I.S. List l boleh kosong, tetapi tidak penuh */
/* F.S. val adalah elemen terakhir l yang baru */

/* ********** MENCARI PASANGAN TITIK TERDEKAT ********** */
/* *** Menggunakan algoritma brute force *** */
void FindClosestPairBF(ListPoint *l, Point *p1, Point *p2, float *min, int *count);
/* *** Menggunakan algoritma divide and conquer *** */
void FindClosestPairDC(ListPoint *l, int size, Point *p1, Point *p2, float *min, int *count);

#endif
```

3.5 ListPoint.c



```c
/* File: ListPoint.c*/
/* Deskripsi: Implementasi ListPoint.h */

#include <stdio.h>
#include "ListPoint.h"

/* Kamus Global */
Point VAL_UNDEF = {.X = -9999,
                   .Y = -9999,
                   .Z = -9999};

/* ********** KONSTRUKTOR ********** */
/* Konstruktor : create list kosong  */
void CreateListPoint(ListPoint *l)
{
    /* KAMUS LOKAL */
    int i;
    /* ALGORITMA */
    for (i = 0; i <= CAPACITY - 1; i++)
    {
        ELMT(*l, i) = VAL_UNDEF;
    }
}
/* Konstruktor : create list random  */
void CreateListPointRandom(ListPoint *l, int size)
{
    /* KAMUS LOKAL */
    int i;
    Point p;
    /* ALGORITMA */
    for (i = 0; i < size; i++)
    {
        p = MakeRandomPoint();
        ELMT(*l, i) = p;
    }
    for (i = size; i <= CAPACITY - 1; i++)
    {
        ELMT(*l, i) = VAL_UNDEF;
    }
}
```



```c
/* ********** SELEKTOR (TAMBAHAN) ********** */
/* *** Banyaknya elemen *** */
int length(ListPoint l)
{
    /* KAMUS LOKAL */
    int i;
    /* ALGORITMA */
    i = 0;
    while ((!Equal(ELMT(l, i), VAL_UNDEF)) && i < CAPACITY)
    {
        i += 1;
    }
    return i;
}

/* ********** TEST KOSONG/PENUH ********** */
/* *** Test List kosong *** */
boolean isEmpty(ListPoint l)
{
    /* KAMUS LOKAL */
    /* ALGORITMA */
    return (length(l) == 0);
}
/* *** Test List penuh *** */
boolean isFull(ListPoint l)
{
    /* KAMUS LOKAL */
    /* ALGORITMA */
    return (length(l) == 1000);
}
```

```
/* ********** SORTING ********** */
void selectionSortX(ListPoint *l, int i, int j)
{
    /* KAMUS LOKAL */
    int k;
    /* ALGORITMA */
    if (i < j)
    {
        partisiX(l, i, j);
        selectionSortX(l, i + 1, j);
    }
}
void partisiX(ListPoint *l, int i, int j)
{
    /* KAMUS LOKAL */
    int k, idxmin;
    Point temp;
    /* ALGORITMA */
    idxmin = i;
    for (k = i + 1; k <= j; k++)
    {
        if (Absis(ELMT(*l, k)) < Absis(ELMT(*l, idxmin)))
        {
            idxmin = k;
        }
    }
    temp = ELMT(*l, i);
    ELMT(*l, i) = ELMT(*l, idxmin);
    ELMT(*l, idxmin) = temp;
}
void selectionSortY(ListPoint *l, int i, int j)
{
    /* KAMUS LOKAL */
    int k;
    /* ALGORITMA */
    if (i < j)
    {
        partisiY(l, i, j);
        selectionSortY(l, i + 1, j);
    }
}
void partisiY(ListPoint *l, int i, int j)
{
    /* KAMUS LOKAL */
    int k, idxmin;
    Point temp;
    /* ALGORITMA */
    idxmin = i;
    for (k = i + 1; k <= j; k++)
    {
        if (Ordinat(ELMT(*l, k)) < Ordinat(ELMT(*l, idxmin)))
        {
            idxmin = k;
        }
    }
    temp = ELMT(*l, i);
    ELMT(*l, i) = ELMT(*l, idxmin);
    ELMT(*l, idxmin) = temp;
}
```

```
/* ********** OPERASI LAIN ********** */
void copyList(ListPoint lIn, int firstIdxCopy, int lastIdxCopy, ListPoint *lOut)
{
    /* KAMUS LOKAL */
    int i, size, offset;
    /* ALGORITMA */
    size = lastIdxCopy - firstIdxCopy + 1;
    for (i = 0; i < size; i++)
    {
        offset = i + firstIdxCopy;
        ELMT(*lOut, i) = ELMT(lIn, offset);
    }
}

/* ********** MENAMBAH ELEMEN DI AKHIR ********** */
/* *** Menambahkan elemen terakhir *** */
void insertLast(ListPoint *l, ElType val)
{
    /* KAMUS LOKAL */
    /* ALGORITMA */
    ELMT(*l, length(*l)) = val;
}

/* ********** MENCARI PASANGAN TITIK TERDEKAT ********** */
/* *** Menggunakan algoritma brute force *** */
void FindClosestPairBF(ListPoint *l, Point *p1, Point *p2, float *min, int *count)
{
    /* KAMUS LOKAL */
    float d;
    /* ALGORITMA */
    *min = 999999999;
    *p1 = ELMT(*l, 0);
    *p2 = ELMT(*l, 1);
    for (int i = 0; i < length(*l); i++)
    {
        for (int j = i + 1; j < length(*l); j++)
        {
            d = Distance(ELMT(*l, i), ELMT(*l, j));
            *count = *count + 1;
            if (d < *min)
            {
                *min = d;
                *p1 = ELMT(*l, i);
                *p2 = ELMT(*l, j);
            }
        }
    }
}
```

```
/* ********** MENCARI PASANGAN TITIK TERDEKAT ********** */
/* *** Menggunakan algoritma brute force *** */
void FindClosestPairBF(ListPoint *l, Point *p1, Point *p2, float *min, int *count)
{
    /* KAMUS LOKAL */
    float d;
    /* ALGORITMA */
    *min = 999999999;
    *p1 = ELMT(*l, 0);
    *p2 = ELMT(*l, 1);
    for (int i = 0; i < length(*l); i++)
    {
        for (int j = i + 1; j < length(*l); j++)
        {
            d = Distance(ELMT(*l, i), ELMT(*l, j));
            *count = *count + 1;
            if (d < *min)
            {
                *min = d;
                *p1 = ELMT(*l, i);
                *p2 = ELMT(*l, j);
            }
        }
    }
}

/* *** Menggunakan algoritma divide and conquer *** */
void FindClosestPairDC(ListPoint *l, int size, Point *p1, Point *p2, float *min, int *count)
{
    /* KAMUS LOKAL */
    float minLeft, minRight;
    ListPoint S1, S2, Sstrip;
    int size1, size2, sizeStrip;
    int Xmid;
    int i, j;
    Point left1, left2, right1, right2;
    int countLeft, countRight;
    float distance01, distance02, distance12;
    float dStrip;
    /* ALGORITMA */
    /* Basis 1: Banyak elemen genap */
    if (size == 2)
    {
        *min = Distance(ELMT(*l, 0), ELMT(*l, 1));
        *p1 = ELMT(*l, 0);
        *p2 = ELMT(*l, 1);
        *count = *count + 1;
    }
```

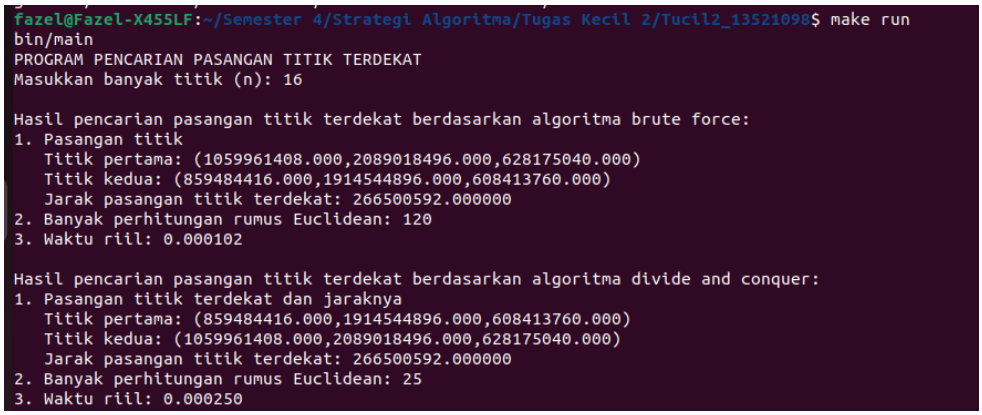
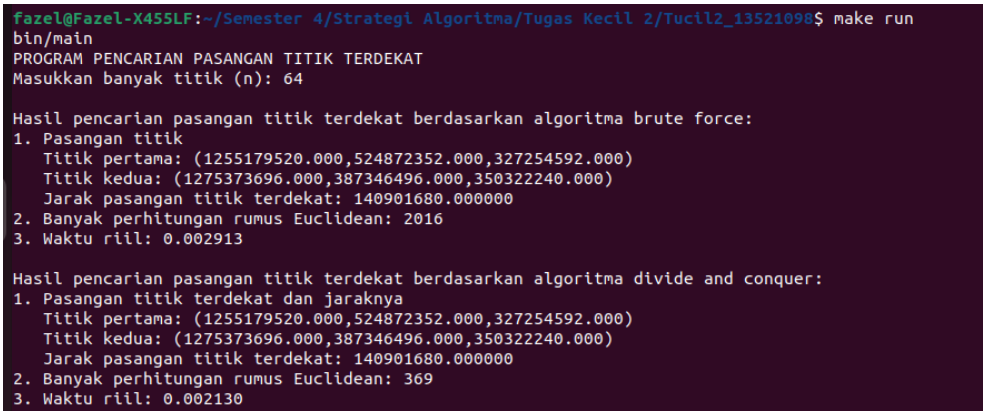
```
    /* Basis 2: Banyak elemen ganjil */
    else if (size == 3)
    {
        /* Hitung jarak setiap pasangan titik */
        distance01 = Distance(ELMT(*l, 0), ELMT(*l, 1));
        *count = *count + 1;
        distance02 = Distance(ELMT(*l, 0), ELMT(*l, 2));
        *count = *count + 1;
        distance12 = Distance(ELMT(*l, 1), ELMT(*l, 2));
        *count = *count + 1;

        /* Mencari pasangan titik terdekat */
        if (distance01 < distance02)
        {
            if (distance01 < distance12) // 01 < 02 && 01 < 12
            {
                *min = distance01;
                *p1 = ELMT(*l, 0);
                *p2 = ELMT(*l, 1);
            }
            else // 12 < 01 < 02
            {
                *min = distance12;
                *p1 = ELMT(*l, 1);
                *p2 = ELMT(*l, 2);
            }
        }
        else
        {
            if (distance02 < distance12) // 02 < 01 && 02 < 12
            {
                *min = distance02;
                *p1 = ELMT(*l, 0);
                *p2 = ELMT(*l, 2);
            }
            else // 12 < 02 < 01
            {
                *min = distance12;
                *p1 = ELMT(*l, 1);
                *p2 = ELMT(*l, 2);
            }
        }
    }
```

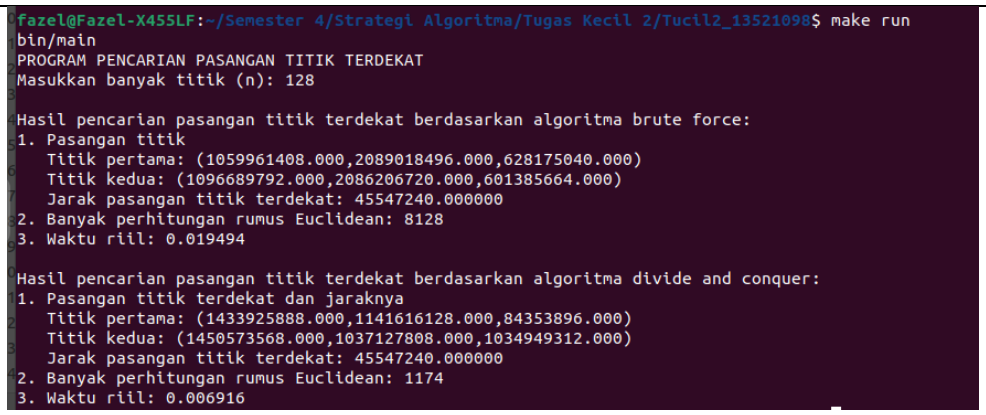
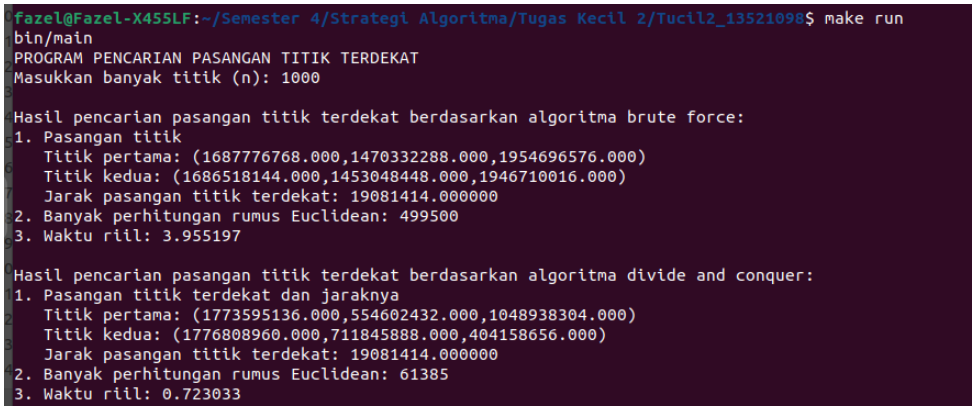
```
    else
    {
        /* Hitung banyak elemen S1 dan S2 */
        size1 = size / 2;
        size2 = size - size1;
        /* Inisialisasi S1 */
        CreateListPoint(&S1);
        copyList(*l, 0, size1 - 1, &S1);
        /* Inisialisasi S2 */
        CreateListPoint(&S2);
        copyList(*l, size1, size - 1, &S2);
        /* Mencari sepasang titik terdekat di S1 dan S2 */
        FindClosestPairDC(&S1, size1, &left1, &left2, &minLeft, &countLeft);
        FindClosestPairDC(&S2, size2, &right1, &right2, &minRight, &countRight);
        /* Menentukan pasangan titik terdekat dari S1 dan S2*/
        if (minLeft < minRight)
        {
            *min = minLeft;
            *p1 = left1;
            *p2 = left2;
        }
        else
        {
            *min = minRight;
            *p1 = right1;
            *p2 = right2;
        }
        *count = countLeft + countRight;

        /* ***Mencari pasangan titik terdekat di Sstrip*** */
        CreateListPoint(&Sstrip);
        /* Mencari absis garis L: Xn/2 */
        Xmid = size / 2;
        /* Mencari semua titik di S1 yang berada di dalam strip */
        for (i = 0; i < length(S1); i++)
        {
            if ((Xmid - Absis(ELMT(S1, i))) <= *min)
            {
                insertLast(&Sstrip, ELMT(S1, i));
            }
        }
        /* Mencari semua titik di S2 yang berada di dalam strip */
        for (i = 0; i < length(S2); i++)
        {
            if ((Absis(ELMT(S2, i)) - Xmid) <= *min)
            {
                insertLast(&Sstrip, ELMT(S2, i));
            }
        }
        /* Mengurutkan titik di Sstrip berdasarkan ordinat menaik */
        selectionSortY(&Sstrip, 0, length(Sstrip) - 1);
        /* Menghitung jarak setiap pasangan titik di Sstrip */
        /* Dibandingkan dengan jarak terdekat sebelumnya  */
        for (i = 0; i < length(Sstrip) - 1; i++)
        {
            for (j = i + 1; j < length(Sstrip) - 1; j++)
            {
                if ((ABS(Absis(ELMT(Sstrip, i)), Absis(ELMT(Sstrip, j))) > *min) || (ABS(Ordinat(ELMT(Sstrip, i)), Ordinat(ELMT(Sstrip, j))) > *min))
                {
                    /* Tidak diproses */
                }
                else
                {
                    dStrip = Distance(ELMT(Sstrip, i), ELMT(Sstrip, j));
                    *count = *count + 1;
                    if (dStrip < *min)
                    {
                        *min = dStrip;
                    }
                }
            }
        }
    }
}
```

BAB IV
PENGUJIAN PROGRAM

Program ini diuji dengan spesifikasi computer sebagai berikut.

1. Processor: Intel Core i5-5005U 2.2Ghz
2. RAM: 4 GB
3. Disk: SSD 256 GB
4. VGA: NVIDIA GT930M
5. OS: Linux Ubuntu 22.04

Masukan	Keluaran
n = 16	
n = 64	

n = 128	 fazel@Fazel-X455LF:~/Semester 4/Strategi Algoritma/Tugas Kecil 2/Tucil2_13521098$ make run bin/main PROGRAM PENCARIAN PASANGAN TITIK TERDEKAT Masukkan banyak titik (n): 128 Hasil pencarian pasangan titik terdekat berdasarkan algoritma brute force: 1. Pasangan titik Titik pertama: (1059961408.000,2089018496.000,628175040.000) Titik kedua: (1096689792.000,2086206720.000,601385664.000) Jarak pasangan titik terdekat: 45547240.000000 2. Banyak perhitungan rumus Euclidean: 8128 3. Waktu riil: 0.019494 Hasil pencarian pasangan titik terdekat berdasarkan algoritma divide and conquer: 1. Pasangan titik terdekat dan jaraknya Titik pertama: (1433925888.000,1141616128.000,84353896.000) Titik kedua: (1450573568.000,1037127808.000,1034949312.000) Jarak pasangan titik terdekat: 45547240.000000 2. Banyak perhitungan rumus Euclidean: 1174 3. Waktu riil: 0.006916
n = 1000	 fazel@Fazel-X455LF:~/Semester 4/Strategi Algoritma/Tugas Kecil 2/Tucil2_13521098$ make run bin/main PROGRAM PENCARIAN PASANGAN TITIK TERDEKAT Masukkan banyak titik (n): 1000 Hasil pencarian pasangan titik terdekat berdasarkan algoritma brute force: 1. Pasangan titik Titik pertama: (1687776768.000,1470332288.000,1954696576.000) Titik kedua: (1686518144.000,1453048448.000,1946710016.000) Jarak pasangan titik terdekat: 19081414.000000 2. Banyak perhitungan rumus Euclidean: 499500 3. Waktu riil: 3.955197 Hasil pencarian pasangan titik terdekat berdasarkan algoritma divide and conquer: 1. Pasangan titik terdekat dan jaraknya Titik pertama: (1773595136.000,554602432.000,1048938304.000) Titik kedua: (1776808960.000,711845888.000,404158656.000) Jarak pasangan titik terdekat: 19081414.000000 2. Banyak perhitungan rumus Euclidean: 61385 3. Waktu riil: 0.723033

DAFTAR PUSTAKA

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

LAMPIRAN

Pranala Github: https://github.com/fazelginanda/Tucil2_13521098

CHECKLIST

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