**LAPORAN TUGAS KECIL II**

**IF2211 STRATEGI ALGORITMA**

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

****

Disusun oleh

Akmal Mahardika Nurwahyu Pratama 13521070

**SEMESTER IV TAHUN 2022/2023**

**JURUSAN TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BAB I**

**ALGORITME DEVIDE AND CONQUER**

Algoritme *devide and conquer* adalah algoritme yang pemograman yang didasarkan pada rekrusif multi-cabang. berdasarkan bahasanya, *devide* artinya membagi dan *conquer* artinya mengatasi. Algoritma ini memiliki langkah :

1. **Divide**

Membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama).

1. **Conquer**

Memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif).

1. **Combine**

Menggabungkan solusi masing-masing masalah sehingga membentuk solusi masalah semula.

Adapun contoh persoalan yang dapat diselesaikan dengan algoritme *devide and conquer* :

1. Persoalan MinMaks (mencari nilai minimum dan nilai maksimum)
2. Menghitung perpangkatan
3. Persoalan pengurutan (sorting) – Mergesort dan Quicksort
4. Mencari sepasang titik terdekat (closest pair problem)
5. Convex Hull
6. Perkalian matriks
7. Perkalian bilangan bulat besar
8. Perkalian dua buah polinom

Pada Tucil 2, mahasiswa diminta mengembangkan algoritma mencari sepasang titik terdekat pada bidang 3D. Misalkan terdapat n buah titik pada ruang 3D. Setiap titik P di dalam ruang dinyatakan dengan koordinat P = (x, y, z). Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titk P1 = (x1, y1, z1) dan P2 = (x2, y2, z2) dihitung dengan rumus Euclidean berikut:



**BAB II**

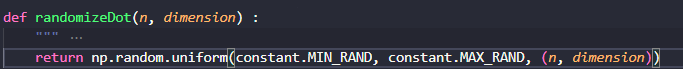
**SOURCE PROGRAM**

Program dibuat dengan menggunakan bahasa python dengan menggunakan *library*

* time
* matplotlib
* numpy
* random
* math

Dengan menggunakan fungsi / prosedur:}

|  |  |
| --- | --- |
| **File tools.py** | |
|  | Membuat array of Titik dengan ukuran n x n. Array 2D diisi dengan nilai random. Nilai elemen dalam array 2d mungkin ada yang sama |
| randomizeDot2(n, dimension) | Sama seperti randomizedot tapi pakai loop |
| isCloser(dot1, dot2, close\_dis) | True jika jarak abs(dot1[i]-dot2[i]) < close\_dis\*\*2 untuk i 0..len(dot1) |
| deanDistance(dot1, dot2) | Menghitung Eucledian Distance antara 2 titik, n Dimensi |
| mergeSort(arr\_dot) | Melakukan merge sort sebuah array of titik berdasarkan nilai x1 |
| euclideanDistance(dot1, dot2) | Menghitung Eucledian Distance antara 2 titik, n Dimensi |



Sebuah gambar berisi teks, cuplikan layar, monitor, layar

Deskripsi dibuat secara otomatis

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

def mergeSort(*arr\_dot*) :

    """ Melakukan merge sort sebuah array of titik berdasarkan nilai x1 """

    if len(*arr\_dot*) > 1:

        # Devide

        mid = len(*arr\_dot*)//2

        L = *arr\_dot*[:mid]

        R = *arr\_dot*[mid:]

        mergeSort(L)

        mergeSort(R)

        # Counquer

        i = j = k = 0

        l\_len = len(L)

        r\_len = len(R)

        temp\_arr = np.zeros((len(*arr\_dot*), len(*arr\_dot*[0])))

        while i < l\_len and j < r\_len :

            if L[i][0] < R[j][0]:

                temp\_arr[k] = L[i]

                i += 1

            else:

                temp\_arr[k] = R[j]

                j += 1

            k += 1

        while i < l\_len :

            temp\_arr[k] = L[i]

            i += 1

            k += 1

        while j < r\_len :

            temp\_arr[k] = R[j]

            j += 1

            k += 1

        for i in range(len(*arr\_dot*)):

*arr\_dot*[i] = temp\_arr[i]

|  |  |
| --- | --- |
| **File DnCTitik.py** | |
| devidenConquer(arr\_dot, nCal) | Mencari jarak terdekat antara 2 titik dengan devide and conquer jarak dicari dari array of titik |
| baseCaseDNC(arr\_dot, nCal) | Basecase dari devidenConquer, Mencari jarak terdekat antara 2 titik dari array 3 atau 2 titik |
| closestPairStrip(left\_arr, right\_arr, close\_dis, close\_dots, mid, nCal) | Mencari Jarak titik terdekat antara 2 titik yang dibatasi garis khayalan berdasarkan divide and conquer |
| stripClose(l\_arr, r\_arr, mid, close\_dis) | Mencari titik terdekat antara 2 titik yang dibatasi garis khayalan berdasarkan divide and conquer |

def devidenConquer(*arr\_dot*, *nCal*) :

    """ Mencari jarak terdekat antara 2 titik dengan devide and conquer

    jarak dicari dari array of titik """

    length\_arr = len(*arr\_dot*)

    if length\_arr <= 3 :

        return baseCaseDNC(*arr\_dot*, *nCal*)

    else :

        mid = length\_arr//2

        left\_arr = *arr\_dot*[:mid]

        right\_arr = *arr\_dot*[mid:]

        dis1, dots1, *nCal* = devidenConquer(left\_arr, *nCal*)

        dis2, dots2, *nCal* = devidenConquer(right\_arr, *nCal*)

        close\_dis = min(dis1, dis2)

        if close\_dis == dis1 :

            close\_dots = dots1

        else :

            close\_dots = dots2

        dis\_mid, dots\_mid, *nCal* = closestPairStrip(left\_arr, right\_arr, close\_dis, close\_dots, *arr\_dot*[mid], *nCal*)

        close\_dis = min(close\_dis, dis\_mid)

        if close\_dis == dis\_mid :

            close\_dots = dots\_mid

        return close\_dis, close\_dots, *nCal*

def baseCaseDNC(*arr\_dot*, *nCal*) :

    """ Basecase dari devidenConquer | Mencari jarak terdekat antara 2 titik dari array 3 atau 2 titik """

    dis = tools.euclideanDistance(*arr\_dot*[0], *arr\_dot*[1])

*nCal* += 1

    dots = [*arr\_dot*[0], *arr\_dot*[1]]

    if (len(*arr\_dot*) == 3) :

        # Cek jika ada 3 titik

        for i in range(2) :     # 0-2 dan 1-2

            temp\_dis = tools.euclideanDistance(*arr\_dot*[i], *arr\_dot*[2])

*nCal* += 1

            if temp\_dis < dis :

                dis = temp\_dis

                dots = [*arr\_dot*[i], *arr\_dot*[2]]

    return dis, dots, *nCal*

def closestPairStrip(*left\_arr*, *right\_arr*, *close\_dis*, *close\_dots*, *mid*, *nCal*) :

    strip\_arr, n\_left, n\_right = stripClose(*left\_arr*, *right\_arr*, *mid*, *close\_dis*)

    dots = *close\_dots*

    dis = *close\_dis*

    # Titik kiri hanayan mengecek titik kanan garis khayalan dan sebaliknya (dengan jarak < close\_dis)

    for i in range(n\_left) :

        for j in range(n\_left, n\_left+n\_right) :

            if (tools.isCloser(strip\_arr[i], strip\_arr[j], dis)) :

                temp\_dis = tools.euclideanDistance(strip\_arr[i], strip\_arr[j])

*nCal* += 1

                if temp\_dis < dis :

                    dis = temp\_dis

                    dots = [strip\_arr[i], strip\_arr[j]]

    return dis, dots, *nCal*

|  |  |
| --- | --- |
| **File ioApp.py** | |
| inputHandle() | Memanggil fungsi/prosedur *input* |
| inputNandDimension() | Menangani input dari pengguna berupa dimensi dan banyak titik |
| outputHandle(min\_distance, min\_dots,nCal, time) | Mencetak hasil perhitungan program |
| printTitik(arr\_dot) | mencetak pasangan titik dalam bentuk (x1, y1,...) dan (x2, y2,...) |
| StartScreen() | Spalsh dari Startscreen |
| BoxOpenScreen(name) | Box pembatas (buka) |
| BoxCloseScreen(name) | Box pembatas (tutup) |

def inputHandle() :

    n, dimension = inputNandDimension()

    # n, dimendion = fileinput()

    return n, dimension

def inputNandDimension():

    """ Menangani input dari pengguna berupa dimensi dan banyak titik]"""

    while (True) :

        n = input("Masukkan jumlah titik  : ")

        try :

            n = int(n)

            if n < 2 :

                print(dc.B\_Red + f"Jumlah titik {dc.Underline}minimal 2!" + dc.Reset)

                continue

            else :

                break

        except ValueError :

            print(dc.B\_Red + "Input harus berupa angka!" + dc.Reset)

            continue

    while (True) :

        dimension = input("Masukkan dimensi titik : ")

        try :

            dimension = int(dimension)

            if dimension < 1 :

                print(dc.B\_Red + f"Dimensi {dimension} tidak terdefini! {dc.Underline}dimensi minimal 1!" + dc.Reset)

            else :

                break

        except ValueError :

            print(dc.B\_Red + "Input harus berupa angka!" + dc.Reset)

            continue

    return n, dimension

def outputHandle(*min\_distance*, *min\_dots*, *nCal*, *time*) :

    """ Mencetak hasil perhitungan program """

    print("Jarak terdekat adalah              :", *min\_distance*)

    print("Pasangan Titik terdekat adalah     :", *end*=' ')

    # Print pasangan titik

    printTitik(*min\_dots*)

    print("Jumlah perhitungan jarak euclidean :", *nCal*)

    print("Waktu perhitungan                  :", *time*, "milidetik")

def printTitik(*arr\_dot*) :

    """ mencetak pasangan titik dalam bentuk (x1, y1,...) dan (x2, y2,...) """

    for i in range(len(*arr\_dot*)) :

        point = *arr\_dot*[i]

        print(f"( {point[0] :.4f}", *end* = "")

        for j in range(1, len(point)):

            print(f", {point[j] :.4f}", *end* = "")

        print(" )", *end* = "")

        if i % 2 == 0:

            print(dc.B\_Green+" dan ", *end* = "" + dc.Reset)

        else :

            print()

def StartScreen() :

    """ Spalsh dari Startscreen """

    print(dc.B\_Magenta,"""

 $$$$$$\ $$\                                      $$\           $$$$$$$\ $$\           $$\

$$  \_\_$$\$$ |                                     $$ |          $$  \_\_$$\\\_\_|          $$ |

$$ /  \\_\_$$ |$$$$$$\  $$$$$$$\ $$$$$$\  $$$$$$$\$$$$$$\         $$ |  $$ $$\ $$$$$$$\$$$$$$\   $$$$$$\ $$$$$$$\  $$$$$$$\ $$$$$$\

$$ |     $$ $$  \_\_$$\$$  \_\_\_\_\_$$  \_\_$$\$$  \_\_\_\_\_\\_$$  \_|        $$ |  $$ $$ $$  \_\_\_\_\_\\_$$  \_|  \\_\_\_\_$$\$$  \_\_$$\$$  \_\_\_\_\_$$  \_\_$$\

$$ |     $$ $$ /  $$ \$$$$$$\ $$$$$$$$ \$$$$$$\   $$ |          $$ |  $$ $$ \$$$$$$\   $$ |    $$$$$$$ $$ |  $$ $$ /     $$$$$$$$ |

$$ |  $$\$$ $$ |  $$ |\\_\_\_\_$$\$$   \_\_\_\_|\\_\_\_\_$$\  $$ |$$\       $$ |  $$ $$ |\\_\_\_\_$$\  $$ |$$\$$  \_\_$$ $$ |  $$ $$ |     $$   \_\_\_\_|

\$$$$$$  $$ \$$$$$$  $$$$$$$  \$$$$$$$\$$$$$$$  | \$$$$  |      $$$$$$$  $$ $$$$$$$  | \$$$$  \$$$$$$$ $$ |  $$ \$$$$$$$\\$$$$$$$\

 \\_\_\_\_\_\_/\\_\_|\\_\_\_\_\_\_/\\_\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\\_\_\_\_\_\_\_/   \\_\_\_\_/       \\_\_\_\_\_\_\_/\\_\_\\_\_\_\_\_\_\_/   \\_\_\_\_/ \\_\_\_\_\_\_\_\\_\_|  \\_\_|\\_\_\_\_\_\_\_|\\_\_\_\_\_\_\_|

          """, dc.Reset)

    print("{: ^144}".format(dc.B\_Yellow + dc.Underline + "Akmal Mahardika Nurwahyu Pratama - 13521070") + dc.Reset + "\n")

def BoxOpenScreen(*name*) :

    """ Box pembatas (buka) """

    print(dc.Bold + "┌{}{:^24}{}┐".format("─"\*53,*name*,"─"\*53) + dc.Reset + "\n")

def BoxCloseScreen(*name*) :

    """ Box pembatas (tutup) """

    print("\n" + dc.Bold + "└{}{:^24}{}┘".format("─"\*53,*name*,"─"\*53) + dc.Reset + "\n")

|  |  |
| --- | --- |
| **File bruteforce.py** | |
| bruteforceDots(arr\_dot) | Mencari pasangan titk terdekat dengan algoritma brute force |

def bruteforceDots(*arr\_dot*) :

    """ Mencari pasangan titk terdekat dengan algoritma brute force """

    nCal = 0

    dis = -1

    close\_dis = 99999

    close\_dots = []

    for i in range(len(*arr\_dot*)) :

        for j in range(i+1, len(*arr\_dot*)) :

            dis = tools.euclideanDistance(*arr\_dot*[i], *arr\_dot*[j])

            nCal += 1

            if dis < close\_dis :

                close\_dis = dis

                close\_dots = [*arr\_dot*[i], *arr\_dot*[j]]

    return close\_dis, close\_dots, nCal

|  |  |
| --- | --- |
| **File visualizer.py** | |
| visualize(arr\_dots, min\_dots) | Visualisasi array of titik |
| OneDPlot(arr\_dots, min\_dots) | Plotting 1D array of Titik |
| TwoDPlot(arr\_dots, min\_dots) | Plotting 2D array of Titik |
| ThreeDPlot(arr\_dots, min\_dots) | Plotting 3D array of Titik |
| connectDots(arr\_dots, dim) | Menghubungkan pasangan titik yang ada di dalam array |

def visualize(*arr\_dots*, *min\_dots*) :

    """ Visualisasi array of Titik """

*arr\_dots* = np.array(*arr\_dots*)

*min\_dots* = np.array(*min\_dots*)

    dim = len(*arr\_dots*[0])

    if dim == 1 :

        OneDPlot(*arr\_dots*, *min\_dots*)

        return True

    elif dim == 2 :

        TwoDPlot(*arr\_dots*, *min\_dots*)

        return True

    elif dim == 3 :

        ThreeDPlot(*arr\_dots*, *min\_dots*)

        return True

    else :

        print(dc.B\_Red + "Tidak dapat melakukan visualisasi" + dc.Reset)

        return False

def OneDPlot(*arr\_dots*, *min\_dots*) :

    """ Plotting 1D array of Titik """

    ax = plt.gca()

    ax.set\_title("Visualisasi Titik")

    plt.plot(*arr\_dots*, np.zeros\_like(*arr\_dots*), 'bo')

    plt.plot(*min\_dots*, np.zeros\_like(*min\_dots*), 'ro')

    connectDots(*min\_dots*,1)

    plt.xlabel('X')

    plt.show()

def TwoDPlot(*arr\_dots*, *min\_dots*) :

    """ Plotting 2D array of Titik """

    ax = plt.gca()

    ax.set\_title("Visualisasi Titik")

    plt.scatter(*arr\_dots*[:,0], *arr\_dots*[:,1], *marker*='o', *color*='b')

    plt.scatter(*min\_dots*[:,0], *min\_dots*[:,1], *marker*='o', *color*='r')

    connectDots(*min\_dots*,2)

    plt.xlabel('X')

    plt.ylabel('Y')

    plt.show()

def ThreeDPlot(*arr\_dots*, *min\_dots*) :

    """ Plotting 3D array of Titik """

    fig = plt.figure()

    ax = fig.add\_subplot(*projection*='3d')

    ax.set\_title("Visualisasi Titik")

    ax.scatter(*arr\_dots*[:,0], *arr\_dots*[:,1], *arr\_dots*[:,2], *color*='b')

    ax.scatter(*min\_dots*[:,0], *min\_dots*[:,1], *min\_dots*[:,2], *color*='r')

    connectDots(*min\_dots*,3, ax)

    ax.set\_xlabel('X')

    ax.set\_ylabel('Y')

    ax.set\_zlabel('Z')

    plt.show()

def connectDots(*arr\_dots*, *dim*, *ax* = None) :

    """ Menghubungkan pasangan titik yang ada di dalam array """

    if *dim* == 1 :

        for i in range(0, len(*arr\_dots*), 2) :

            plt.plot([*arr\_dots*[i], *arr\_dots*[i+1]], [0,0], 'r-')

    elif *dim* == 2 :

        for i in range(0, len(*arr\_dots*), 2) :

            plt.plot([*arr\_dots*[i,0], *arr\_dots*[i+1,0]], [*arr\_dots*[i,1], *arr\_dots*[i+1,1]], 'r-')

    elif *dim* == 3 :

        # fig = plt.figure()

        # ax = fig.add\_subplot(projection='3d')

        for i in range(0, len(*arr\_dots*), 2) :

*ax*.plot([*arr\_dots*[i,0], *arr\_dots*[i+1,0]], [*arr\_dots*[i,1], *arr\_dots*[i+1,1]], [*arr\_dots*[i,2], *arr\_dots*[i+1,2]], 'r-')

Adapun file lain berisi atribut-atribut yang melengkapi program, beberapa diantaranya

* File constant.py

# declare constant

MAX\_RAND = 500

MIN\_RAND = -500

* File designCli.py

# Text Colour

Black = "\u001b[0;30m"

Red = "\u001b[0;31m"

Green = "\u001b[0;32m"

Yellow = "\u001b[0;33m"

Blue = "\u001b[0;34m"

Magenta = "\u001b[0;35m"

Cyan = "\u001b[0;36m"

White = "\u001b[0;37m"

Reset = "\u001b[0;00m"

B\_Black = "\u001b[30;1m"

B\_Red = "\u001b[31;1m"

B\_Green = "\u001b[32;1m"

B\_Yellow = "\u001b[33;1m"

B\_Blue = "\u001b[34;1m"

B\_Magenta = "\u001b[35;1m"

B\_Cyan = "\u001b[36;1m"

B\_White = "\u001b[37;1m"

Reset = "\u001b[0m"

# bg colour

Bg\_Black = "\u001b[40m"

Bg\_Red = "\u001b[41m"

Bg\_Green = "\u001b[42m"

Bg\_Yellow = "\u001b[43m"

Bg\_Blue = "\u001b[44m"

Bg\_Magenta = "\u001b[45m"

Bg\_Cyan = "\u001b[46m"

Bg\_White = "\u001b[47m"

# Text Style

Bold = "\u001b[1m"

Underline = "\u001b[4m"

Reversed = "\u001b[7m"

File -file diatas membentuk main program sebagai berikut (main.py)

# File : main.py

# Application to find nearest 2 point from n point in 3D space and calculate distance between them

import time

import bruteforce as bf

import tools as tl

import DnCTitik as DnC

import ioApp as io

import visualizer as vis

if \_\_name\_\_ == "\_\_main\_\_":

    io.StartScreen()

    n, dimension = io.inputHandle()

    arr\_dots = tl.randomizeDot2(n, dimension)

    tl.mergeSort(arr\_dots)

    dnc\_nCal = 0

    dnc\_timeS = time.time()

    dnc\_min\_dis, dnc\_min\_dots, dnc\_nCal  = DnC.devidenConquer(arr\_dots, dnc\_nCal)

    dnc\_timeE = time.time()

    io.BoxOpenScreen("Devide and Conquer")

    io.outputHandle(dnc\_min\_dis, dnc\_min\_dots, dnc\_nCal, (dnc\_timeE - dnc\_timeS)\*1000)

    io.BoxCloseScreen("Devide and Conquer")

    vis.visualize(arr\_dots, dnc\_min\_dots)

    bf\_nCal = 0

    bf\_timeS = time.time()

    bf\_min\_dis, bf\_min\_dots, bf\_nCal  = bf.bruteforceDots(arr\_dots)

    bf\_timeE = time.time()

    io.BoxOpenScreen("Brute Force")

    io.outputHandle(bf\_min\_dis, bf\_min\_dots, bf\_nCal, (bf\_timeE - bf\_timeS)\*1000)

    io.BoxCloseScreen("Brute Force")

    vis.visualize(arr\_dots, bf\_min\_dots)

**BAB III**

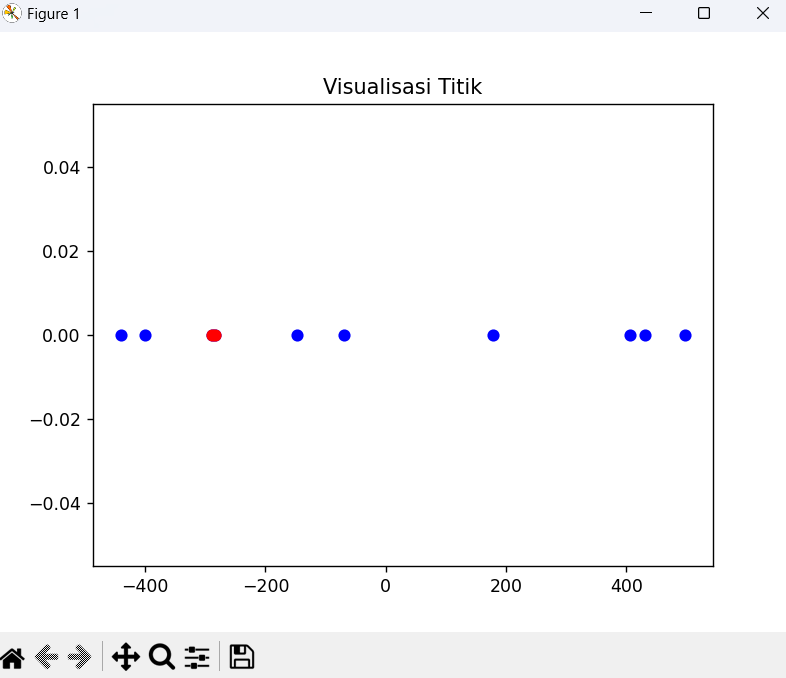
**TEST PROGRAM**

Input program terdiri dari 2 yaitu, jumlah titik (n) dan dimensi titik (d). Berikut merupakan beberapa test program yang dilakukan, test program dilakukan pada prosesor Intel i7-11567G dengan 8 GB Memori :

1. Test input keyboard benar untuk n = 10 dan d = 1

Sebuah gambar berisi teks

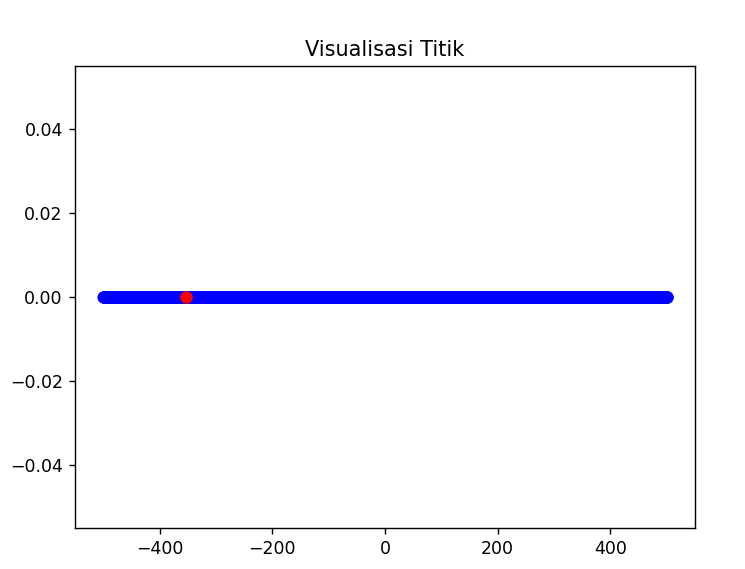
Deskripsi dibuat secara otomatis



1. Test input keyboard benar untuk n = 5000 dan d = 1

Sebuah gambar berisi teks

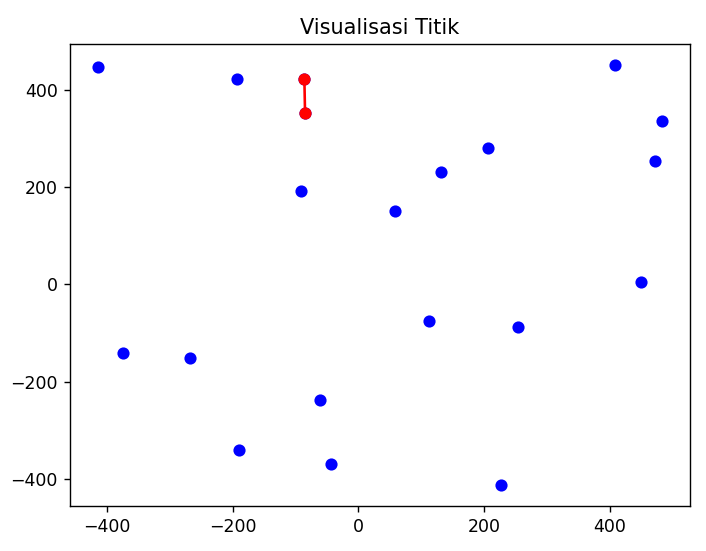
Deskripsi dibuat secara otomatis



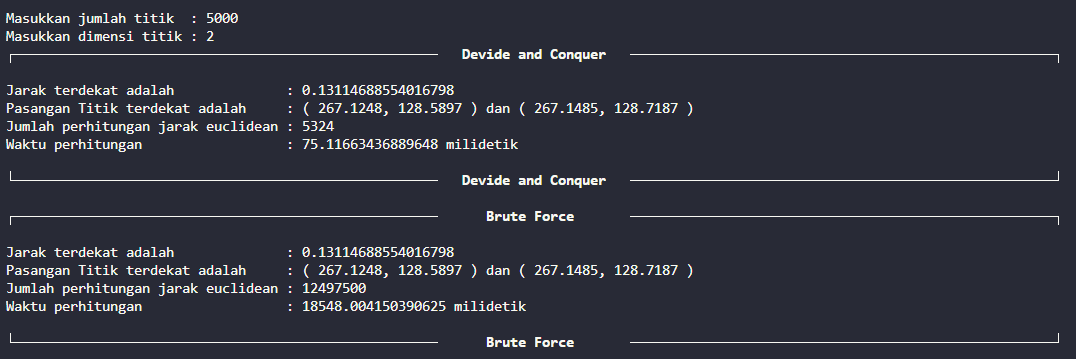
1. Test input keyboard benar untuk n = 20 dan d = 2

Sebuah gambar berisi teks, monitor, cuplikan layar, dalam ruangan

Deskripsi dibuat secara otomatis



1. Testinput keyboard untuk n = 5000 dan d = 5



Sebuah gambar berisi teks, elektronik, etalase, cuplikan layar

Deskripsi dibuat secara otomatis

1. Test *handling Input* (*Input* Salah)

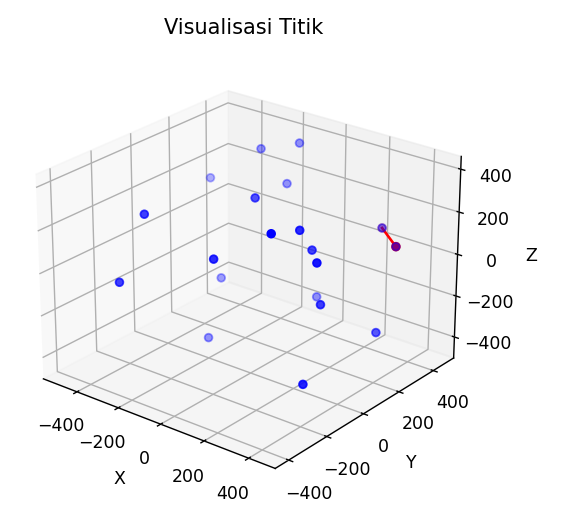
Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

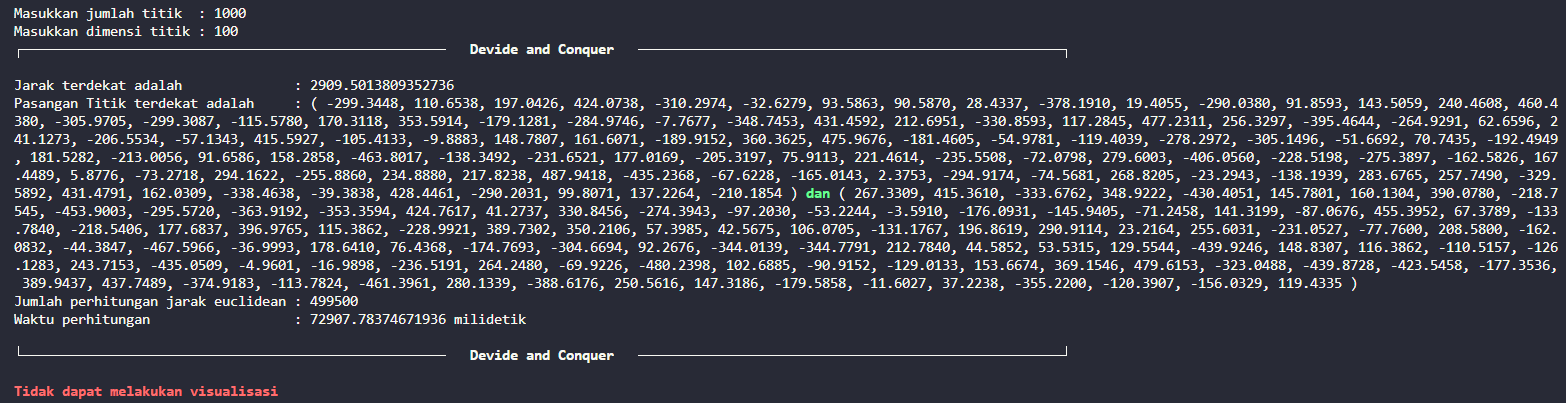
1. Test keyboard untuk n = 30 dan d = 3

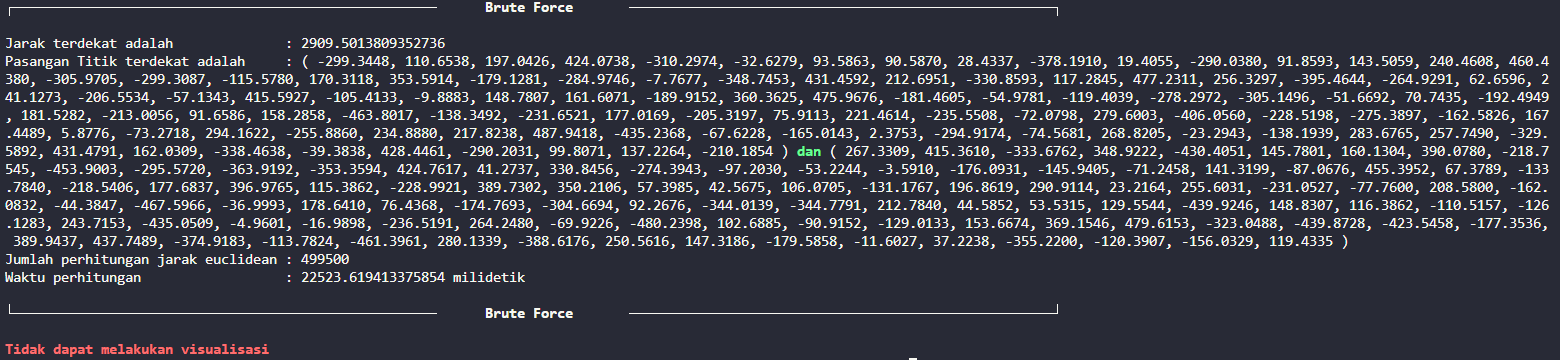
Sebuah gambar berisi teks, monitor, cuplikan layar, layar

Deskripsi dibuat secara otomatis



1. Test keyboard input untuk n = 1000 dan d = 100

****

****

**BAB IV**

**KESIMPULAN DAN SARAN**

Saat membuat program penulis menyadari beberapa hal dari algoritma devide and conquer untuk perhitunggan jarak 2 titik terdekat, yaitu :

1. Algoritme tidak selalu efektif, jika setelah membagi terdapat banyak titik yang dekat dengan ’garis khayalan’ akan memerlukan brute force yang lama.
2. Tidak dapat dipungkiri bahwa algoritme ini tetap memerlukan brute force ketika terjadi kejadian seperti poin 1.

**LAMPIRAN**

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| Program berhasil dikompilasi tanpa ada kesalahan. | ✓ |  |
| Program berhasil running | ✓ |  |
| Program dapat menerima masukan dan dan menuliskan luaran. | ✓ |  |
| Luaran program sudah benar (solusi closest pair benar) | ✓ |  |
| Bonus 1 dikerjakan | ✓ |  |
| Bonus 2 dikerjakan | ✓ |  |

Repository Github

<https://github.com/akmaldika/Tucil2_13521070.git>