

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



Disusun Oleh:

Melvin Kent Jonathan	13521052
Angela Livia Arumsari	13521094

INSTITUT TEKNOLOGI BANDUNG
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
2022/2023

DAFTAR ISI

IF2211 - STRATEGI ALGORITMA

DAFTAR ISI

BAB I - DESKRIPSI MASALAH	1
BAB II - METODE Pencarian Titik Terdekat dalam R^n	2
2.1 Algoritma Brute Force	2
2.2 Algoritma Divide and Conquer	2
2.3 Penerapan Algoritma Brute Force untuk Mencari Titik Terdekat dalam R^n	2
2.4 Penerapan Algoritma Divide and Conquer untuk Mencari Titik Terdekat dalam R^n	3
BAB III - IMPLEMENTASI PROGRAM Pencarian Titik Terdekat dalam R^n	5
BAB IV - PENGUJIAN	17
4.1 Contoh Kasus	17
4.2 Analisis Pengujian	29
BAB V - PENUTUP	31
5.1 Kesimpulan	31
Lampiran	32

BAB I

DESKRIPSI MASALAH

Mencari sepasang titik terdekat dengan Algoritma *Divide and Conquer* sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Pada Tugil 2 kali ini, kami 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 titik $P_1 = (x_1, y_1, z_1)$ dan $P_2 = (x_2, y_2, z_2)$ dihitung dengan rumus Euclidean berikut:

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

Buatlah program dalam Bahasa C/C++/Java/Python/Golang/Ruby/Perl (pilih salah satu) untuk mencari sepasang titik yang jaraknya terdekat satu sama lain dengan menerapkan algoritma *divide and conquer* untuk penyelesaiannya, dan perbandingannya dengan Algoritma *Brute Force*.

Masukan program:

- n
- titik-titik (dibangkitkan secara acak) dalam koordinat (x, y, z)

Luaran program:

- sepasang titik yang jaraknya terdekat dan nilai jaraknya
- banyaknya operasi perhitungan rumus Euclidean
- waktu riil dalam detik (spesifikasikan komputer yang digunakan)
- Bonus 1 (Nilai = 7,5) penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya.
- Bonus 2 (nilai = 7,5): Generalisasi program anda sehingga dapat mencari sepasang titik terdekat untuk sekumpulan vektor di R_n , setiap vektor dinyatakan dalam bentuk $x = (x_1, x_2, \dots, x_n)$

BAB II

METODE PENCARIAN TITIK TERDEKAT DALAM R^n

2.1 Algoritma *Brute Force*

Algoritma *brute force* merupakan sebuah strategi algoritma yang memberikan solusi atau penyelesaian dari suatu persoalan dengan mencoba seluruh kemungkinan yang ada. Algoritma *brute force* efektif dalam mencari solusi yang tepat dari sebuah persoalan, tetapi cenderung tidak efisien ketimbang strategi-strategi algoritma lainnya. Strategi algoritma ini membandingkan sebuah solusi dari suatu persoalan dengan seluruh kemungkinan kandidat solusi lainnya dan mencari solusi manakah yang tepat dan sesuai dengan kriteria yang diinginkan oleh pemrogram. Oleh karena itu, algoritma *brute force* cocok untuk dijadikan pembanding atau penguji coba algoritma lainnya karena dapat menghasilkan solusi yang tepat. Pada program yang kami buat, algoritma *brute force* akan dipakai sebagai salah satu opsi penyelesaian pencarian pasangan titik terdekat dalam R^n sekaligus untuk membandingkan kebenaran solusi yang diperoleh dengan algoritma *divide and conquer*.

2.2 Algoritma *Divide and Conquer*

Algoritma *divide and conquer* merupakan sebuah strategi algoritma yang terus membagi persoalan ke dalam beberapa sub-persoalan yang serupa dengan ukuran yang lebih kecil hingga diperoleh sebuah persoalan basis yang dapat diselesaikan dengan sederhana. Diperolehnya solusi atas sebuah sub-persoalan diharapkan dapat mempermudah atau menurunkan tingkat kerumitan pencarian solusi dari persoalan induknya. Oleh karena sifatnya yang berulang, pengimplementasian algoritma *divide and conquer* sangat natural untuk dituangkan dalam skema rekursif. Strategi algoritma ini dapat dipakai dalam menyelesaikan persoalan pencarian pasangan titik terdekat dalam R^n karena persoalan ini dapat diselesaikan dengan terus membagi himpunan titik ke dalam sub-himpunan titik yang lebih kecil serta menyelesaikan persoalan dengan dimensi ruang yang lebih tinggi. Pada program yang kami buat, algoritma *divide and conquer* akan dipakai sebagai metode utama penyelesaian persoalan.

2.3 Penerapan Algoritma *Brute Force* untuk Mencari Titik Terdekat dalam R^n

Berikut ini adalah langkah yang diterapkan oleh algoritma *brute force* dalam mencari titik terdekat dalam R^n :

1. Iterasikan perhitungan jarak untuk setiap kemungkinan pasangan titik pada himpunan titik yang ada.
2. Simpan jarak pasangan titik pertama sebagai jarak terdekat sementara dan simpan koordinat dari pasangan titik tersebut.
3. Apabila terdapat jarak antara dua buah titik yang lebih kecil daripada jarak terdekat sementara, perbarui nilai jarak terdekat sementara dengan mengganti nilainya menjadi

nilai jarak antara dua buah titik yang baru saja dites. Perbarui juga koordinat pasangan titik terdekat.

4. Iterasikan langkah ketiga hingga seluruh kemungkinan perhitungan jarak dapat selesai dilakukan.

2.4 Penerapan Algoritma *Divide and Conquer* untuk Mencari Titik Terdekat dalam R^n

Algoritma Divide and Conquer yang digunakan untuk mencari titik terdekat dalam R^n merupakan pengembangan dari algoritma yang digunakan pada dimensi dua. Pada dimensi dua, tahapan pencarian titik terdekat adalah sebagai berikut.

1. Titik diurutkan menaik berdasarkan nilai absisnya.
2. Jika jumlah titik hanya dua, maka langsung hitung jarak kedua titik tersebut.
3. Lakukan tahap divide dengan membagi himpunan titik menjadi dua himpunan, yaitu S_1 dan S_2 dengan jumlah titik yang sama. Pembagian himpunan titik ini dihampiri oleh titik tengah pada himpunan titik yang sudah terurut.
4. Tahap conquer merupakan tahap rekursif memanggil metode divide and conquer terhadap masing-masing himpunan titik.
5. Lakukan pencarian titik pada strip selebar $2d$ di sekitar garis tengah dengan d jarak terdekat titik yang di dapat dari kedua himpunan titik. Jarak absis dan ordinat titik pada strip ini maksimal adalah d agar jarak titik tidak lebih besar dari d .
6. Himpunan titik di dalam strip diurutkan berdasarkan ordinat sehingga ketika jarak ordinat sudah melebihi d , perhitungan jarak titik dapat dihentikan. Dengan restriksi tersebut, maksimal perhitungan jarak untuk tiap titik pada himpunan strip adalah enam.
7. Lakukan tahap combine dengan membandingkan jarak terdekat titik dari yang di dapat pada himpunan kanan dan kiri sebelumnya dengan jarak terdekat titik dari himpunan titik di dalam strip.

Untuk mencari titik terdekat untuk dimensi yang lebih tinggi, diterapkan penyesuaian terhadap algoritma pencarian titik terdekat pada dimensi dua. Pada dimensi yang lebih tinggi, algoritma divide and conquer akan dipanggil kembali untuk himpunan titik pada strip. Pada algoritma ini, tiap rekursi divide and conquer untuk himpunan titik di dalam strip akan melakukan perubahan sumbu yang dianggap sebagai absis. Misalnya, pada dimensi tiga, pemanggilan divide and conquer pada himpunan titik strip akan menjadikan sumbu y seakan-akan sebagai absis dan sumbu z seakan-akan sebagai ordinat.

Dengan algoritma ini, berapapun nilai dimensi yang digunakan, penghitungan titik terdekat menggunakan dimensi dua dapat terus berlaku. Beberapa penyesuaian pada algoritma dari algoritma pencarian titik terdekat dimensi dua yang diterapkan yaitu sebagai berikut.

1. Himpunan titik yang diterima akan diurutkan berdasarkan dimensi titik dikurangi dimensi rekursi. Misalnya pada rekursi pertama titik akan diurutkan berdasarkan sumbu x ,

sedangkan pada dimensi rekursi yang telah berkurang titik akan diurutkan berdasarkan sumbu y.

2. Jika jumlah titik kurang dari empat, maka dapat langsung dilakukan pemanggilan metode brute force.
3. Setelah menemukan himpunan titik di dalam strip, algoritma akan mengecek dimensi rekursi saat itu. Jika dimensi rekursi belum dua, maka akan dilakukan pemanggilan ulang terhadap fungsi divide and conquer dengan dimensi rekursi berkurang satu. Namun, jika dimensi rekursi sudah dua, akan dilakukan pencarian titik terdekat sesuai dengan algoritma pada dimensi dua.

Dengan pendekatan seperti di atas, divide and conquer akan sangat optimal untuk perhitungan jarak terdekat titik pada dimensi yang tidak terlalu tinggi. Pada dimensi yang sangat tinggi, rekursi yang dilakukan akan banyak pula yang membuat waktu algoritma bisa menjadi lebih lambat dari brute force.

BAB III

IMPLEMENTASI PROGRAM PENCARIAN TITIK TERDEKAT DALAM R^n

Berikut merupakan implementasi algoritma yang kami terapkan pada program.

Pengimplementasian algoritma *brute force* dan algoritma *divide and conquer* terdapat pada file:

- BruteForce.py
- DivideAndConquer.py

BruteForce.py

```
def bruteForce(points, ctr):
    if (len(points) <= 1):
        return None, None, ctr

    closestPairs = []
    for i in range(len(points)):
        for j in range(i+1, len(points)):
            if (i == 0 and j ==1):
                shortestDistance, ctr =
Utils.getDistancePoints(points[i], points[j], ctr)
                closestPairs += [[points[i], points[j]]]
            else :
                distPoint, ctr = Utils.getDistancePoints(points[i],
points[j], ctr)
                if (distPoint < shortestDistance):
                    closestPairs = []
                    shortestDistance = distPoint
                    closestPairs = [[points[i], points[j]]]
                elif (distPoint == shortestDistance):
                    closestPair = [points[i], points[j]]
                    closestPairs = Utils.appendIfNotSame(closestPairs,
closestPair)

    return closestPairs, shortestDistance, ctr
```

DivideAndConquer.py

```
# Quicksort function to sort points based on index
# Points is an array of points, index specify the index of coordinates
to be sorted on
def quickSortPoints(points, index):
    if len(points) <= 1:
        return points
    else:
        pivot = points[0]
        left = []
        right = []
        for point in points[1:]:
            if point[index] < pivot[index]:
                left.append(point)
            else:
                right.append(point)
        return quickSortPoints(left, index) + [pivot] +
quickSortPoints(right, index)

# Recursive function to find closest pair of points in the stripe with
dim dimensions
def findStripClosest(points, dist, dim, ctr):
    # Initialize closest distance
    closestDist = dist
    # Compute the closest dist only if the dimension is 2
    if (dim <= 2):
        if (len(points) <= 1):
            return None, None, ctr
        newStripPoints = points
        closestPairs = []
        if (dim == 2):
            newStripPoints = quickSortPoints(points, dim - 1)
        n = len(newStripPoints)
        # max_inter= 0
        for i in range (n):
```



```

        # inter = 0
        for j in range (i+1, n):
            if (newStripPoints[j][dim-1] - newStripPoints[j][dim-1]
> closestDist):
                break
            distPoint, ctr = getDistancePoints(newStripPoints[i],
newStripPoints[j], ctr)
            # inter+= 1
            if (distPoint < closestDist):
                closestPairs = []
                closestDist = distPoint
                closestPair = [newStripPoints[i], newStripPoints[j]]
                closestPairs = [closestPair]
            elif (distPoint == closestDist):
                closestPair = [newStripPoints[i], newStripPoints[j]]
                closestPairs = appendIfNotSame(closestPairs,
closestPair)

        # max_inter = inter if inter > max_inter else max_inter
        # print(max_inter)
        # If the dimension is not two
        else:
            closestPairs, closestDist, ctr = findClosestPair(points,
len(points), dim-1, ctr)
        return closestPairs, closestDist, ctr

def findClosestPair(unsortedPoints, n: int, dim: int, ctr: int):
    if (n <= 3):
        closestPairs, closestDist, ctr = bruteForce(unsortedPoints, ctr)
    else:
        # Sort points
        points = quickSortPoints(unsortedPoints, len(unsortedPoints[0])
- dim)

        # Divide phase
        mid = n // 2
        midPoint = points[mid][len(unsortedPoints[0]) - dim]
        leftPart = points[0:mid]

```

```

        rightPart = points[mid:]

        leftClosestPoints, distLeftPair, ctr = findClosestPair(leftPart,
len(leftPart), dim, ctr)
        rightClosestPoints, distRightPair, ctr =
findClosestPair(rightPart, len(rightPart), dim, ctr)
        closestPairs = []

        if (distLeftPair < distRightPair):
            closestPairs = leftClosestPoints
            closestDist = distLeftPair
        elif (distRightPair < distLeftPair):
            closestPairs = rightClosestPoints
            closestDist = distRightPair
        else:
            closestPairs = leftClosestPoints
            closestPairs = extendIfNotSame(closestPairs,
rightClosestPoints)
            closestDist = distLeftPair

        # The closesDist from two halves became the delta
        leftBound = midPoint - closestDist
        rightBound = midPoint + closestDist

        filteredPoints = [x for x in points if leftBound <=
x[len(unsortedPoints[0]) - dim] <= rightBound]
        stripClosestPoints, distStripPair, ctr =
findStripClosest(filteredPoints, closestDist, dim, ctr)

        if (stripClosestPoints != None):
            if (distStripPair < closestDist):
                closestPairs = []
                closestPairs = stripClosestPoints
                closestDist = distStripPair
            elif (distStripPair == closestDist):
                closestPairs = extendIfNotSame(closestPairs,
stripClosestPoints)

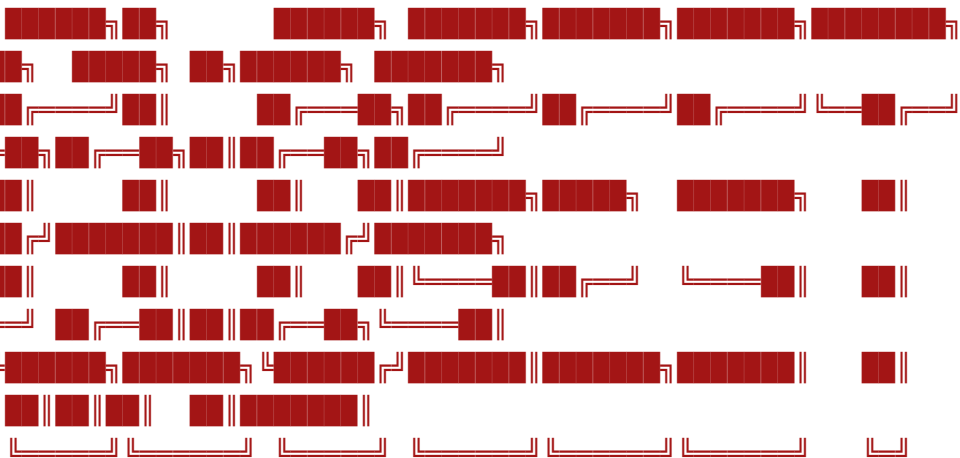
        return closestPairs, closestDist, ctr

```

Adapun fungsionalitas penopang program dituangkan ke dalam file:

- IO.py
- Utils.py

IO.py

```
def splashScreen():  
    print("\033[32mWelcome to closest pair program :D\033[00m\n")  
    print("""\033[34m  
  
\033[00m""")  
  
def mainMenu():  
    print("""  
=====
```

```
\033[31m*****\033[00m\033[34m MENU  
\033[00m\033[31m*****\033[00m  
=====
```

```
1. Generate Random Points  
2. File Input  
3. Exit  
""")  
  
# def algorithmMenu():  
#     print("""  
#         Choose the algorithm to find closest pair:  
#         1. Divide and Conquer  
#         2. Brute Force
```

```

# """

def saveConfirmation(nPoints, nDims, points, closestPairsDNC,
closestDistDNC, elapsed_timeDNC, ctrOptDNC, closestPairsBF,
closestDistBF, elapsed_timeBF, ctrOptBF):
    save = input("\nDo you want to save the result to a .txt file (Y/N)?
")

    while (save != "Y" and save!= "y" and save!= "N" and save != "n"):
        print("Your input is not valid. Please enter a valid input!")
        save = input("Do you want to save the result to a .txt file
(Y/N)? ")

    if (save == "Y" or save == "y") :
        fileName = input("The result will be saved to a file. Enter your
desired file name: ")
        writeToFile(nPoints, nDims, fileName, points, closestPairsDNC,
closestDistDNC, elapsed_timeDNC, ctrOptDNC, closestPairsBF,
closestDistBF, elapsed_timeBF, ctrOptBF)

def writePairsToFile(f, pairs):
    f.write("Closest Pair(s) of Points: \n")
    for i in range (len(pairs)):
        f.write("\t" + str(i+1) + ". " + Utils.formatPoint(pairs[i][0])
+ " - " + Utils.formatPoint(pairs[i][1]) + "\n")

def writeToFile(nPoints, nDims, fileName, points, closestPairsDNC,
closestDistDNC, elapsed_timeDNC, ctrOptDNC, closestPairsBF,
closestDistBF, elapsed_timeBF, ctrOptBF):
    dir_path =
os.path.dirname(os.path.dirname(os.path.realpath(__file__)))
    f = open(dir_path + "\\output\\" + fileName + ".txt", "w")
    f.write("Number of Points           : " + str(nPoints)+ "\n")
    f.write("Dimension                   : " + str(nDims)+ "\n")
    f.write("Points                           : " + str(points) + "\n")

    # Result by Divide and Conquer
    f.write("\n===== DIVIDE AND CONQUER =====\n")
    writePairsToFile(f, closestPairsDNC)

```

```

        f.write("Closest Distance                : " + str(closestDistDNC) +
"\n")
        f.write("Number of Euclidean Operations : " + str(ctrOptDNC)+ "\n")
        f.write("Execution Time                : " + str(elapsed_timeDNC) +
" ms\n")

    # Result by Brute Force
    f.write("\n===== BRUTE FORCE =====\n")
    writePairsToFile(f, closestPairsBF)
    f.write("Closest Distance                : " + str(closestDistBF) +
"\n")
    f.write("Number of Euclidean Operations : " + str(ctrOptBF)+ "\n")
    f.write("Execution Time                : " + str(elapsed_timeBF) +
" ms\n")
    f.close()

def inputHandler():
    while True:
        try:
            nPoints = int(input("Enter the n points you want to
generate: "))
        except:
            print("Your input of n points is not valid. Please input a
number with a minimum of two!")
            continue
        if (nPoints >= 2):
            break

        print("Your input of n points is not valid. Please input a
number with a minimum of two!")
        while True:
            try:
                nDims = int(input("Enter the d dimension of points you want
to generate: "))
            except:
                print("Your input of n points is not valid. Please input a
number with a minimum of two!")
                continue

            if (nDims > 0):

```

```

        break
        print("Your input of d dimensions is not valid. Please input a
number with a minimum of one!")
        return nPoints, nDims

def readFromFile(fileName):
    dir_path =
os.path.dirname(os.path.dirname(os.path.realpath(__file__)))
    f = open(dir_path + "\\test\\" + fileName + ".txt", "r")
    lines = f.readlines()
    points = [list(map(int, line.strip().split())) for line in lines]
    print(points)
    f.close()
    return points

def visualize(points, pairs) :
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')

    for x, y, z in points:
        isPair = False
        i = 0
        color = ['blue', 'green', 'cyan', 'magenta', 'yellow', 'black',
'tab:orange', 'tab:olive']
        while (not isPair and i < len(pairs)):
            if(x == (pairs[i][0][0]) and y == pairs[i][0][1] and z ==
pairs[i][0][2]) or (x == pairs[i][1][0] and y == pairs[i][1][1] and z ==
pairs[i][1][2]):
                isPair = True
            else :
                i += 1
        if (isPair):
            ax.scatter(x,y,z, c=color[(i%8)], marker='o')
        else :
            ax.scatter(x,y,z, c='r', marker='o')

```

```

plt.show()

def printPairs(pairs):
    print("Closest Pair(s) of Points:")
    for i in range (len(pairs)):
        print("\t" + str(i+1) + ". \033[94m" +
Utils.formatPoint(pairs[i][0]) + "\033[00m - \033[93m" +
Utils.formatPoint(pairs[i][1]) + "\033[00m")

```

Utils.py

```

def getDistancePoints(point1, point2, ctrOpt):
    dist = 0
    for i in range (len(point1)):
        dist += (point1[i] - point2[i]) ** 2

    ctrOpt+=1;
    return math.sqrt(dist), ctrOpt

def appendIfNotSame(closestPairs, pointPair):
    has_common_element = False
    for arr in closestPairs:
        if (pointPair == [[]]):
            has_common_element = True
            break
        if ((pointPair[0] == arr[0] or pointPair[0] == arr[1]) and
(pointPair[1] == arr[0] or pointPair[1] == arr[1])):
            has_common_element = True
            break

    # Melakukan append hanya jika pasangan point pada pointPair tidak
    sama dengan salah satu pasangan pada closestPairs
    if not has_common_element:
        closestPairs.append(pointPair)
    return closestPairs

def extendIfNotSame(closestPairs, pointPairs):

```

```

    for arr in pointPairs:
        closestPairs = appendIfNotSame(closestPairs, arr)

    return closestPairs

def formatPoint(point):
    strPoint = "("
    i = 0
    while (i < len(point)-1):
        strPoint = strPoint + str(point[i]) + " , "
        i += 1

    strPoint = strPoint + str(point[i]) + ")"

    return strPoint

def generateRandomPoints(total_points, total_dim) :
    points = [[0 for j in range (total_dim)] for i in
range(total_points)]

    for i in range (total_points) :
        for j in range (total_dim) :
            points[i][j] = round(random.uniform(-100, 100),2)

    print(points)
    return points

```

Selanjutnya, program utama diimplementasikan pada file **main.py**

main.py

```

def executeDivideAndConquer(inputPoints):
    ctrOpt = 0
    start = time.time()
    closestPairs, closestDist, ctrOpt = findClosestPair(inputPoints,
len(inputPoints), len(inputPoints[0]), ctrOpt)
    end = time.time()
    elapsed_time = (end - start) * 1000 # in milliseconds

```



```

        IO.printPairs(closestPairs)
        print("Closest Distance           :\033[92m", closestDist,
"\033[00m")
        print("Number of Euclidean Operations :\033[91m", ctrOpt,
"\033[00m")
        print("Execution Time           :\033[35m", elapsed_time,
"ms\033[00m")
        if (len(inputPoints[0]) == 3):
            IO.visualize(inputPoints, closestPairs)

        return closestPairs, closestDist, elapsed_time, ctrOpt

def executeBruteForce(inputPoints):
    ctrOpt = 0
    points = inputPoints
    start = time.time()
    closestPairs, closestDist, ctrOpt = bruteForce(points, ctrOpt)
    end = time.time()
    elapsed_time = (end - start) * 1000 # in milliseconds
    IO.printPairs(closestPairs)
    print("Closest Distance           :\033[92m", closestDist,
"\033[00m")
    print("Number of Euclidean Operations :\033[91m", ctrOpt,
"\033[00m")
    print("Execution Time           :\033[35m", elapsed_time,
"ms\033[00m")

    if (len(inputPoints[0]) == 3):
        IO.visualize(inputPoints, closestPairs)

    return closestPairs, closestDist, elapsed_time, ctrOpt

def main():
    IO.splashScreen()
    IO.mainMenu()
    opt = int(input("Enter your choice: "))
    while (opt != 3):
        if (opt == 1 or opt == 2):
            if (opt == 1):

```

```

        nPoints, nDims = IO.inputHandler()
        print("Here are your random points: \n")
        points = Utils.generateRandomPoints(nPoints, nDims)

    elif (opt == 2):
        fileName = input("Enter your file name without .txt: ")
        points = IO.readFromFile(fileName)
        nPoints, nDims = len(points) , len (points[0])

    # Divide and Conquer
    print("\n\033[31m===== \033[00m\033[36mDIVIDE AND
CONQUER\033[00m\033[31m =====\033[00m")
    closestPairsDNC, closestDistDNC, elapsed_timeDNC, ctrOptDNC
= executeDivideAndConquer(points)

    # Brute Froce
    print("\n\033[96m===== \033[00m\033[31mBRUTE
FORCE\033[00m\033[36m =====\033[00m")
    closestPairsBF, closestDistBF, elapsed_timeBF, ctrOptBF =
executeBruteForce(points)

    # Saving Result to A File
    IO.saveConfirmation(nPoints, nDims, points, closestPairsDNC,
closestDistDNC, elapsed_timeDNC, ctrOptDNC, closestPairsBF,
closestDistBF, elapsed_timeBF, ctrOptBF)

    else:
        print("\033[31mInvalid input! \033[00m \n")

    IO.mainMenu()
    opt = int(input("Enter your choice: "))


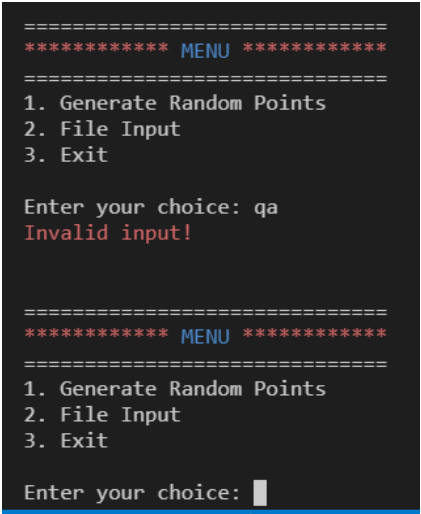
if __name__=="__main__":
    main()

```

BAB IV PENGUJIAN

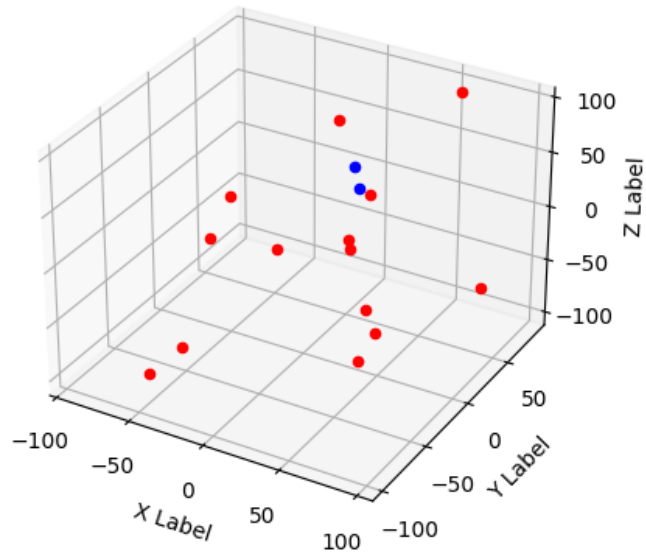
4.1 Contoh Kasus

Berikut ini adalah hasil uji coba program dengan beberapa kasus percobaan yang dijalankan pada laptop Windows 11 dengan prosesor Intel i3-8250 dan RAM 8GB.

Keterangan	Hasil
Inisiasi Program	
Input Validation	<p style="text-align: center;">Input Choice of Menu</p>  <p style="text-align: center;">Input n Points</p>

	<pre> ===== ***** MENU ***** ===== 1. Generate Random Points 2. File Input 3. Exit Enter your choice: 1 Enter the n points you want to generate: 0 Your input of n points is not valid. Please input a number with a minimum of two! Enter the n points you want to generate: 11 </pre> <p style="text-align: center;">Input n Dimensions</p> <pre> Enter the d dimension of points you want to generate: 0 Your input of d dimensions is not valid. Please input a number with a minimum of one! Enter the d dimension of points you want to generate: 3 </pre> <p style="text-align: center;">Input Save File Confirmation</p> <pre> Do you want to save the result to a .txt file (Y/N)? a Your input is not valid. Please enter a valid input! </pre>
Saving Result to A File	<p style="text-align: center;">Seluruh hasil percobaan di bawah telah disimpan pada file yang dapat diakses pada repository.</p> <div style="background-color: #2e3436; color: #eeeeec; padding: 10px; border: 1px solid #2e3436;"> <p>▼ output</p> <ul style="list-style-type: none"> ≡ tc1.txt ≡ tc2.txt ≡ tc3.txt ≡ tc4.txt ≡ tc5.txt ≡ tc6.txt ≡ tc7.txt ≡ tc8.txt ≡ tc9.txt ≡ tc10.txt ≡ tc11.txt </div>

	<pre> output > tc11.txt 1 Number of Points : 16 2 Dimension : 3 3 Points : [[3, 5, 4], [1, 0, 1], [15, 4, 3], [3, 2, 4], [2, 2, 2], [3, 3, 3], [4, 4, 4]] 4 5 ===== DIVIDE AND CONQUER ===== 6 ✓ Closest Pair(s) of Points: 7 1. (3 , 2 , 4) - (3 , 3 , 3) 8 2. (4 , 4 , 4) - (3 , 5 , 4) 9 Closest Distance : 1.4142135623730951 10 Number of Euclidean Operations : 53 11 Execution Time : 0.9908676147460938 ms 12 13 ===== BRUTE FORCE ===== 14 ✓ Closest Pair(s) of Points: 15 1. (3 , 5 , 4) - (4 , 4 , 4) 16 2. (3 , 2 , 4) - (3 , 3 , 3) 17 Closest Distance : 1.4142135623730951 18 Number of Euclidean Operations : 120 19 Execution Time : 0.0 ms 20 </pre>
<p>TC 1 Random n = 16, dim = 3</p>	<pre> ===== ***** MENU ***** ===== 1. Generate Random Points 2. File Input 3. Exit Enter your choice: 1 Enter the n points you want to generate: 16 Enter the d dimension of points you want to generate: 3 Here are your random points: [[-19.54, 2.92, -24.56], [44.15, 6.63, -79.5], [-57.34, -43.13, -98.68], [7.86, 46.17, 33.2], [0.88, 38.65, 77.33], [16.39, 48.77, 8.83], [7.45, 39.52, -30.21], [89.88, -85.74, 29.77], [-51.11, 3.2, 11.99], [-90.95, 42.29, -70.32], [53.51, -28.57, -75.79], [-44.44, -97.39, -76.01], [-5.56, 64.37, -62.14], [18.29, 33.17, 26.31], [57.09, 84.5, 94.46], [96.36, 38.01, -39.59]] ===== DIVIDE AND CONQUER ===== Closest Pair(s) of Points: 1. (18.29 , 33.17 , 26.31) - (7.86 , 46.17 , 33.2) Closest Distance : 18.034882866267807 Number of Euclidean Operations : 75 Execution Time : 0.9992122650146484 ms ===== BRUTE FORCE ===== Closest Pair(s) of Points: 1. (7.86 , 46.17 , 33.2) - (18.29 , 33.17 , 26.31) Closest Distance : 18.034882866267807 Number of Euclidean Operations : 120 Execution Time : 0.0 ms </pre>



TC 2

Random
n = 32, dim = 3

```
===== MENU =====
***** MENU *****
=====
1. Generate Random Points
2. File Input
3. Exit

Enter your choice: 1
Enter the n points you want to generate: 32
Enter the d dimension of points you want to generate: 3
Here are your random points:

[[2.9, -4.23, -4.95], [-87.28, -64.22, -13.47], [16.79, 29.84, -19.72], [95.46, 12.56,
93.91], [25.05, 42.04, -74.69], [-6.39, -84.02, -45.58], [4.24, 22.45, -38.62], [-3.5
3, 65.51, 77.96], [24.01, -65.36, -41.54], [8.96, 40.59, -36.49], [-87.15, -54.69, -50
.12], [-35.74, 19.34, -35.33], [38.23, -87.13, 66.81], [-65.34, -40.16, -88.14], [73.3
7, -32.11, -51.05], [16.42, -70.1, -51.44], [46.62, -37.32, -6.66], [93.58, 57.84, -36
.52], [62.46, 92.7, 86.73], [93.02, 4.62, 85.64], [12.82, 69.18, -49.0], [54.54, -36.7
8, -68.81], [-4.49, 27.24, -15.84], [-28.52, 83.58, -81.83], [-48.69, -52.98, -66.53],
[-80.14, 25.28, 38.76], [-35.28, 19.58, -83.03], [5.25, -18.6, -99.2], [-25.98, -86.7
9, 30.0], [33.68, -63.42, -79.21], [15.71, -73.29, 0.56], [-13.92, -85.41, 47.56]]

===== DIVIDE AND CONQUER =====
Closest Pair(s) of Points:
1. (93.02 , 4.62 , 85.64) - (95.46 , 12.56 , 93.91)
Closest Distance : 11.721352311060356
Number of Euclidean Operations : 119
Execution Time : 2.991199493408203 ms

===== BRUTE FORCE =====
Closest Pair(s) of Points:
1. (95.46 , 12.56 , 93.91) - (93.02 , 4.62 , 85.64)
Closest Distance : 11.721352311060356
Number of Euclidean Operations : 496
Execution Time : 1.0013580322265625 ms
```


TC 3

Random

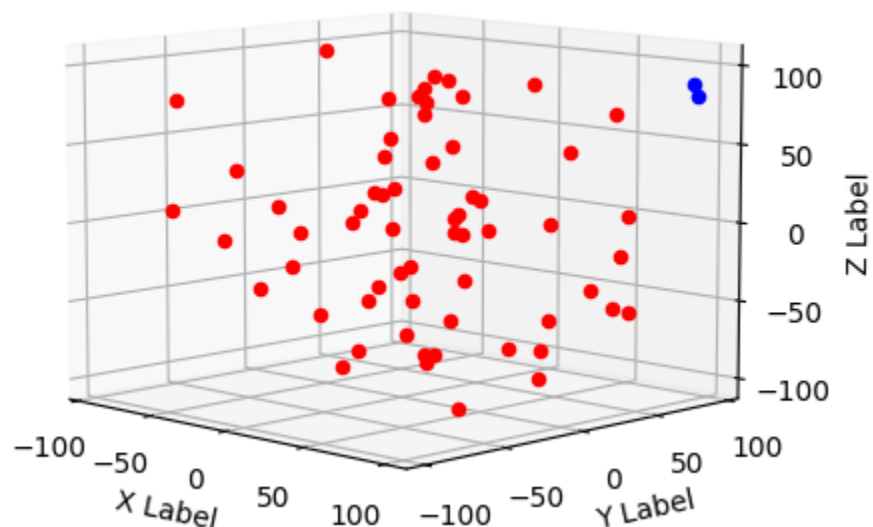
$n = 64$, $\text{dim} = 3$

```
Enter your choice: 1
Enter the n points you want to generate: 1
Your input of n points is not valid. Please input a number with a minimum of two!
Enter the n points you want to generate: 64
Enter the d dimension of points you want to generate: 3
Here are your random points:

[[18.95, -47.5, -70.44], [54.64, 84.11, -61.03], [33.07, 64.04, -69.13], [97.79, -63.8
9, -89.41], [-14.41, 50.29, -7.65], [19.21, -70.56, -44.63], [57.65, 31.2, -95.95], [1
8.79, 52.52, -87.89], [26.66, 64.86, -89.35], [63.43, -57.51, -30.87], [57.71, 92.54,
-64.75], [58.23, 50.44, 45.25], [-8.69, -7.26, -41.3], [-43.51, 85.31, -26.96], [8.11,
-97.42, -25.77], [66.7, -48.53, 53.41], [52.26, 71.86, -47.33], [-24.76, 40.6, 78.21]
, [95.92, -59.59, 98.41], [-4.1, 55.97, 5.41], [45.01, -70.91, 23.49], [-64.01, 63.67,
-52.91], [99.11, 93.22, 80.32], [-91.91, 69.65, -77.4], [-69.98, -78.85, 9.3], [-49.5
5, -65.57, -9.58], [-23.79, 19.25, 15.01], [71.38, -57.55, -62.23], [30.81, 25.6, -5.1
9], [0.81, 40.33, -44.52], [-89.98, -57.58, 73.55], [-71.68, 89.04, 57.69], [-3.95, 36
.3, -69.91], [-58.04, 69.66, 64.95], [-22.64, -43.64, -3.96], [-63.95, 80.0, 50.8], [8
3.7, 61.91, 6.56], [45.38, 51.69, -2.98], [-33.19, -5.86, -98.56], [98.33, 90.46, 87.9
5], [1.77, -9.6, 54.13], [-12.76, 34.96, -93.66], [65.41, -82.8, 39.76], [-26.29, 16.9
8, 73.44], [-75.42, 63.72, 23.38], [85.32, 55.03, -18.21], [-30.69, 69.84, -9.46], [8.
22, 7.7, -91.06], [94.33, 42.26, 75.12], [-32.12, -47.39, 12.41], [39.08, 46.63, 86.45
], [36.48, 8.09, 20.46], [-37.01, 24.91, 8.92], [-90.29, 40.08, 95.61], [49.59, -47.46
, -55.76], [-36.79, -69.32, 37.0], [-5.84, 36.89, 85.19], [-79.04, 86.06, -55.21], [-4
7.41, 41.39, -17.25], [-33.99, -38.1, -28.4], [16.54, 4.83, 94.27], [73.0, -41.11, 9.8
5], [-36.21, 71.49, 34.64], [-21.12, -11.84, -1.19]]

===== DIVIDE AND CONQUER =====
Closest Pair(s) of Points:
1. (98.33 , 90.46 , 87.95) - (99.11 , 93.22 , 80.32)
Closest Distance : 8.15125143766282
Number of Euclidean Operations : 392
Execution Time : 5.031347274780273 ms

===== BRUTE FORCE =====
Closest Pair(s) of Points:
1. (99.11 , 93.22 , 80.32) - (98.33 , 90.46 , 87.95)
Closest Distance : 8.15125143766282
Number of Euclidean Operations : 2016
Execution Time : 5.001306533813477 ms
```



TC 4

Random

$n = 1000$, $\text{dim} = 3$

===== DIVIDE AND CONQUER =====

Closest Pair(s) of Points:

1. $(-87.19, -24.92, 16.37) - (-86.77, -24.59, 16.27)$

Closest Distance : 0.5434151267677437

Number of Euclidean Operations : 13731

Execution Time : 173.99859428405762 ms

===== BRUTE FORCE =====

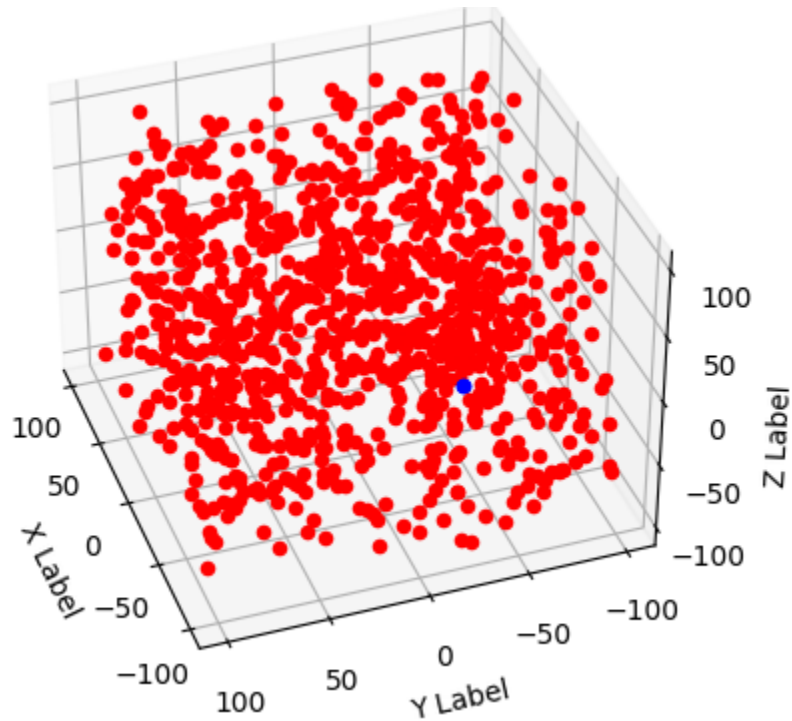
Closest Pair(s) of Points:

1. $(-87.19, -24.92, 16.37) - (-86.77, -24.59, 16.27)$

Closest Distance : 0.5434151267677437

Number of Euclidean Operations : 499500

Execution Time : 1079.9946784973145 ms



TC 5

Random

$n = 64$, $\text{dim} = 1$

```
=====
***** MENU *****
=====
1. Generate Random Points
2. File Input
3. Exit

Enter your choice: 1
Enter the n points you want to generate: 64
Enter the d dimension of points you want to generate: `
Your input of n points is not valid. Please input a number with a minimum of two!
Enter the d dimension of points you want to generate: 1
Here are your random points:

[[-51.53], [-45.56], [-48.05], [47.96], [24.22], [-32.27], [5.11], [-38.25], [76.6], [-87.29], [-53.64], [-51.05], [-46.78], [-33.83], [-96.41], [-27.86], [-42.44], [-86.32], [-21.67], [-15.66], [32.86], [18.13], [18.38], [-10.08], [23.18], [-15.19], [39.05], [-96.88], [44.21], [-47.36], [-70.2], [-90.93], [-2.06], [51.48], [-55.75], [22.35], [97.24], [15.67], [12.36], [3.39], [-66.45], [58.0], [45.53], [87.42], [-29.54], [76.42], [-64.75], [88.11], [62.88], [-12.29], [-8.68], [-42.48], [29.8], [-91.43], [66.35], [80.74], [-62.06], [77.69], [85.5], [-94.56], [51.83], [-85.11], [83.83], [48.06]]

===== DIVIDE AND CONQUER =====
Closest Pair(s) of Points:
1. (-42.48) - (-42.44)
Closest Distance : 0.03999999999999915
Number of Euclidean Operations : 49
Execution Time : 3.0052661895751953 ms

===== BRUTE FORCE =====
Closest Pair(s) of Points:
1. (-42.44) - (-42.48)
Closest Distance : 0.03999999999999915
Number of Euclidean Operations : 2016
Execution Time : 6.032705307006836 ms
```

TC 6

Random

$n = 64$, $\text{dim} = 2$

```
=====
***** MENU *****
=====
1. Generate Random Points
2. File Input
3. Exit

Enter your choice: 1
Enter the n points you want to generate: 64
Enter the d dimension of points you want to generate: 2
Here are your random points:

[[-96.09, 69.34], [-37.34, -8.11], [5.61, -97.89], [-99.78, -66.8], [-91.91, 92.96], [-6.68, -52.43], [86.68, -22.78], [-51.23, 80.61], [-6.5, -17.05], [64.11, -32.8], [-52.72, -96.59], [-1.86, -93.11], [-63.1, 98.03], [-26.63, 51.96], [46.56, 24.52], [-42.87, -73.49], [-6.13, 38.57], [28.05, -3.17], [55.92, -42.09], [27.44, -72.08], [-41.95, -34.59], [-66.09, 25.82], [84.78, -59.82], [15.82, 99.45], [69.57, -91.98], [38.32, -45.95], [29.87, -79.77], [-50.42, 34.67], [-59.15, -94.88], [99.53, -78.75], [57.24, 11.38], [50.94, 37.16], [-87.43, 91.45], [-89.97, -96.21], [-78.0, -58.52], [77.8, 65.36], [-66.62, -33.84], [52.94, 77.93], [22.64, -7.4], [-30.34, 49.12], [-23.53, 58.5], [17.49, 43.1], [-60.71, -70.42], [-50.51, 62.29], [37.47, 50.39], [66.27, -36.5], [71.42, -81.75], [94.36, 84.61], [73.2, 86.17], [-27.18, -37.06], [23.8, -73.21], [-22.41, 59.36], [91.07, 27.13], [53.34, -98.07], [47.78, 67.12], [-36.63, 87.86], [53.26, -53.28], [-78.28, 45.57], [-19.15, 37.52], [18.44, 98.87], [-65.39, -30.35], [64.93, -92.47], [90.86, -52.79], [49.69, 69.73]]

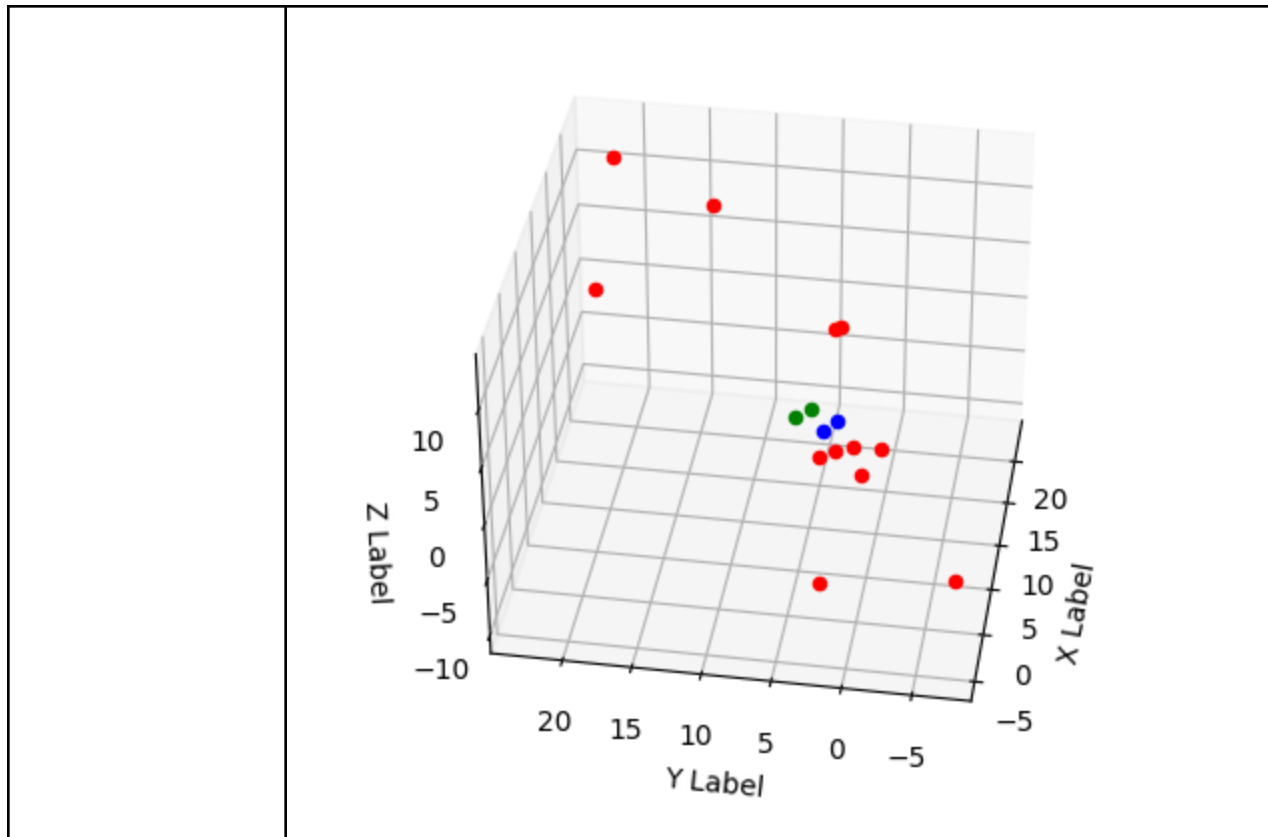
===== DIVIDE AND CONQUER =====
Closest Pair(s) of Points:
1. (-23.53 , 58.5) - (-22.41 , 59.36)
Closest Distance : 1.4120906486483087
Number of Euclidean Operations : 232
Execution Time : 2.9981136322021484 ms

===== BRUTE FORCE =====
Closest Pair(s) of Points:
1. (-23.53 , 58.5) - (-22.41 , 59.36)
Closest Distance : 1.4120906486483087
Number of Euclidean Operations : 2016
Execution Time : 5.999088287353516 ms
```

<p>TC 7</p> <p>Random</p> <p>n = 64, dim = 4</p>	<pre> Enter the d dimension of points you want to generate: 4 Here are your random points: [[69.0, 90.25, -98.46, 69.93], [78.18, -25.99, 52.98, -87.18], [37.5, -95.22, -88.17, -40.24], [-79.47, -1.22, 13.77, -46.39], [36.99, -10.44, -67.09, 91.22], [-63.14, -97. 49, 81.14, 38.16], [54.54, -82.5, -62.31, 52.56], [99.32, -9.93, -34.93, 35.36], [76.1 2, -93.55, 34.21, 9.82], [70.96, -85.21, -98.83, 89.19], [57.41, -83.44, 4.56, 83.84], [62.8, 10.93, -17.32, 68.02], [45.81, -2.67, 27.26, 18.94], [86.2, 47.93, 5.0, -44.23], [-12.17, -55.75, 27.54, 63.64], [-12.79, 95.71, 30.55, 25.11], [-36.96, -90.83, 81. 96, 18.81], [-22.0, -34.05, -15.64, 32.93], [-36.98, 12.77, -62.22, 2.09], [-26.37, -6 6.87, -53.64, -36.15], [45.84, -77.32, -42.62, 74.61], [24.91, -72.6, 67.99, -43.01], [6.08, 13.3, -0.25, 77.19], [50.8, -88.18, 38.73, -44.77], [-28.07, 38.22, 57.05, -59. 49], [42.32, -50.12, 47.48, 40.92], [4.84, 74.65, 58.56, 24.89], [-12.57, -94.32, -54. 24, -34.24], [-96.97, -86.5, -74.24, -25.26], [85.77, -61.54, 30.92, 47.19], [-48.12, -67.35, 43.79, -86.37], [-90.02, 68.87, -57.23, 99.96], [42.31, -10.62, -58.81, -22.57], [-32.04, -64.52, 98.53, -82.6], [8.3, 3.69, 73.52, 47.22], [-48.73, -63.69, 5.87, - 77.67], [-26.35, 55.57, 25.61, -90.31], [37.19, 86.07, 94.5, -91.73], [82.9, -87.32, - 28.64, -42.13], [10.95, 58.27, -48.27, 46.81], [-69.59, -21.69, -72.01, -49.25], [88.9 4, 44.36, 40.7, 91.71], [-22.65, -27.3, 0.48, -36.78], [-91.85, 19.7, 97.1, 49.7], [-7 1.79, -33.15, 90.65, 89.13], [24.67, -80.77, 10.22, -44.01], [68.34, -8.92, -20.49, -7 9.93], [-9.48, 56.0, 80.06, -81.46], [69.03, -82.91, -43.75, -19.0], [-5.81, -70.22, 5 0.86, 90.55], [-19.29, 21.87, 96.67, 64.21], [-41.14, 70.29, -76.39, -62.23], [-56.31, -17.03, 39.36, -24.75], [-69.2, 93.67, -90.0, 29.7], [-83.55, -63.96, 6.04, -80.51], [-25.42, -2.75, -8.53, 6.5], [28.89, 62.79, -43.55, -73.9], [77.18, -22.79, 22.58, 33. 48], [-52.66, -40.03, -37.74, -97.01], [46.11, 88.13, -58.04, -84.48], [-81.68, 46.46, -59.98, -23.0], [-22.48, 71.73, 75.17, 72.55], [-11.13, 57.85, 4.53, 75.65], [58.61, -78.68, -29.17, 43.75]] ===== DIVIDE AND CONQUER ===== Closest Pair(s) of Points: 1. (-12.57 , -94.32 , -54.24 , -34.24) - (-26.37 , -66.87 , -53.64 , -36.15) Closest Distance : 30.788806407524138 Number of Euclidean Operations : 919 Execution Time : 14.03498649597168 ms ===== BRUTE FORCE ===== Closest Pair(s) of Points: 1. (-26.37 , -66.87 , -53.64 , -36.15) - (-12.57 , -94.32 , -54.24 , -34.24) Closest Distance : 30.788806407524138 Number of Euclidean Operations : 2016 Execution Time : 4.037141799926758 ms </pre>
<p>TC 8</p> <p>Random</p> <p>n = 64, dim = 5</p>	<pre> ===== DIVIDE AND CONQUER ===== Closest Pair(s) of Points: 1. (61.62 , 67.65 , -24.53 , 79.97 , 45.19) - (83.15 , 78.38 , -25.79 , 95.48 , 38.73) Closest Distance : 29.369254331698656 Number of Euclidean Operations : 1319 Execution Time : 7.0018768310546875 ms ===== BRUTE FORCE ===== Closest Pair(s) of Points: 1. (83.15 , 78.38 , -25.79 , 95.48 , 38.73) - (61.62 , 67.65 , -24.53 , 79.97 , 45.19) Closest Distance : 29.369254331698656 Number of Euclidean Operations : 2016 Execution Time : 4.028797149658203 ms </pre>

<p>TC 9</p> <p>Random</p> <p>n = 64, dim = 6</p>	<pre> ===== DIVIDE AND CONQUER ===== Closest Pair(s) of Points: 1. (-36.28 , 3.88 , 80.93 , 71.68 , -22.52 , 50.84) - (-26.83 , 17.16 , 52.75 , 38.13 , -34.07 , 47.62) Closest Distance : 48.261233925377425 Number of Euclidean Operations : 3671 Execution Time : 19.036054611206055 ms ===== BRUTE FORCE ===== Closest Pair(s) of Points: 1. (-36.28 , 3.88 , 80.93 , 71.68 , -22.52 , 50.84) - (-26.83 , 17.16 , 52.75 , 38.13 , -34.07 , 47.62) Closest Distance : 48.261233925377425 Number of Euclidean Operations : 2016 Execution Time : 4.304409027099609 ms </pre>
<p>TC 10</p> <p>Random</p> <p>n = 20, dim = 10</p>	<pre> Enter your choice: 1 Enter the n points you want to generate: 20 Enter the d dimension of points you want to generate: 10 Here are your random points: [[-55.28, -51.28, -33.26, -43.95, 89.35, 34.36, -36.77, -69.58, -14.71, 70.95], [23.51 , 94.26, -79.78, 99.92, -13.25, 26.87, -36.53, 23.61, -0.22, -72.21], [-16.34, 89.94, 46.7, -90.75, 5.67, 22.9, -5.98, -14.25, 88.01, -23.47], [30.29, 8.67, -89.64, 62.44, 86.68, -96.26, -17.03, 48.58, 28.52, -61.48], [-37.27, -0.59, -60.84, -19.87, 67.6, 73 .6, -11.04, 39.38, -83.05, -87.27], [59.51, 51.72, 75.3, 72.36, 13.66, -93.95, -5.21, 73.65, 4.63, -82.73], [-66.49, 24.41, -2.63, -30.18, 98.37, -60.06, 3.59, 69.49, 94.15 , -97.56], [74.98, -5.53, 15.0, 28.95, 39.09, -0.83, 81.14, 53.76, -75.76, -12.12], [7 1.83, 88.92, -88.79, 8.7, 4.33, 47.19, -25.92, -57.34, 11.47, -21.56], [-15.26, 0.46, -7.73, 98.82, -24.16, 12.54, 84.29, -65.81, 40.15, -45.96], [-97.43, 30.95, -23.58, -1 7.69, 33.78, 64.23, -4.91, -70.02, 68.21, 95.46], [-60.23, 35.01, -63.58, -15.03, -96. 95, -76.76, 63.23, -52.44, 48.07, -15.79], [65.46, -75.93, 89.23, -80.55, 6.95, 57.58, 69.66, -74.43, -93.69, -43.83], [-39.33, -94.35, -87.56, 86.32, -20.37, -75.71, 75.02 , 68.27, -28.07, -35.3], [52.37, 0.22, 1.19, 66.02, -51.07, -98.53, -89.23, 5.81, -96. 69, 68.87], [-0.58, -94.65, 3.14, 62.21, -67.28, -9.33, 91.32, 72.18, 20.45, 77.98], [69.54, 2.94, -24.82, -6.08, -80.95, 86.0, -66.53, -58.26, -54.8, 41.19], [87.75, 96.79 , 38.51, -18.37, -1.68, 62.62, -17.96, 6.53, -3.08, -47.92], [26.01, 16.52, 63.95, 39. 49, 99.25, 52.71, -33.47, -20.62, -1.12, 51.6], [-87.7, -46.21, 92.87, 39.42, -10.22, -86.26, 42.64, -90.95, -0.19, 84.08]] ===== DIVIDE AND CONQUER ===== Closest Pair(s) of Points: 1. (71.83 , 88.92 , -88.79 , 8.7 , 4.33 , 47.19 , -25.92 , -57.34 , 11.47 , -2 1.56) - (23.51 , 94.26 , -79.78 , 99.92 , -13.25 , 26.87 , -36.53 , 23.61 , -0.22 , -7 2.21) Closest Distance : 144.4130136102699 Number of Euclidean Operations : 5428 Execution Time : 31.999588012695312 ms ===== BRUTE FORCE ===== Closest Pair(s) of Points: 1. (23.51 , 94.26 , -79.78 , 99.92 , -13.25 , 26.87 , -36.53 , 23.61 , -0.22 , -2 1.56) - (71.83 , 88.92 , -88.79 , 8.7 , 4.33 , 47.19 , -25.92 , -57.34 , 11.47 , -2 1.56) Closest Distance : 144.4130136102699 Number of Euclidean Operations : 190 Execution Time : 0.0 ms </pre>

<p>TC 11</p> <p>Baca dari 2d.txt n = 10, dim = 2</p> <p>Terdapat lebih dari 1 pasang titik terdekat.</p>	<pre> ===== ***** MENU ***** ===== 1. Generate Random Points 2. File Input 3. Exit Enter your choice: 2 Enter your file name without .txt: 2d [[77, 22], [43, 52], [41, 24], [34, 24], [27, 24], [23, 2], [24, 15], [12, 54], [22, 54], [41, 14]] ===== DIVIDE AND CONQUER ===== Closest Pair(s) of Points: 1. (34 , 24) - (41 , 24) 2. (27 , 24) - (34 , 24) Closest Distance : 7.0 Number of Euclidean Operations : 26 Execution Time : 0.0 ms ===== BRUTE FORCE ===== Closest Pair(s) of Points: 1. (41 , 24) - (34 , 24) 2. (34 , 24) - (27 , 24) Closest Distance : 7.0 Number of Euclidean Operations : 45 Execution Time : 1.0027885437011719 ms </pre>
<p>TC 12</p> <p>Baca dari 2d.txt n = 10, dim = 2</p> <p>Terdapat lebih dari 1 pasang titik terdekat.</p>	<pre> ===== ***** MENU ***** ===== 1. Generate Random Points 2. File Input 3. Exit Enter your choice: 2 Enter your file name without .txt: 3d [[3, 5, 4], [1, 0, 1], [15, 4, 3], [3, 2, 4], [2, 2, 2], [3, 3, 3], [4, 4, 4], [20, 23, 1], [23, 22, 11], [15, 13, 13], [18, 4, 1], [4, 1, 1], [4, -1, 1], [2, 3, -10], [-5, 2, 7], [0, -7, -7]] ===== DIVIDE AND CONQUER ===== Closest Pair(s) of Points: 1. (3 , 2 , 4) - (3 , 3 , 3) 2. (4 , 4 , 4) - (3 , 5 , 4) Closest Distance : 1.4142135623730951 Number of Euclidean Operations : 53 Execution Time : 0.9908676147460938 ms ===== BRUTE FORCE ===== Closest Pair(s) of Points: 1. (3 , 5 , 4) - (4 , 4 , 4) 2. (3 , 2 , 4) - (3 , 3 , 3) Closest Distance : 1.4142135623730951 Number of Euclidean Operations : 120 Execution Time : 0.0 ms </pre>



4.2 Analisis Pengujian

Pada **hampir** seluruh pengujian yang dilakukan (**kecuali Test Case 9 dan Test Case 10, akan dibahas lebih lanjut**), dapat diketahui bahwa algoritma *divide and conquer* memberikan jumlah operasi euclidean yang lebih sedikit ketimbang yang dilakukan oleh algoritma *brute force*. Namun di sisi lain, algoritma *divide and conquer* memakan waktu eksekusi yang **tidak** melulu lebih sedikit dibandingkan dengan algoritma *brute force*. Hal ini membuktikan bahwa tingkat kompleksitas dari suatu algoritma tidak selalu tercermin dalam waktu aktual eksekusi, melainkan pada jumlah operasi yang dilakukannya relatif terhadap n (jumlah data). Dengan hasil pengujian yang diperoleh, dapat dikatakan bahwa algoritma *divide and conquer* menawarkan alternatif penyelesaian yang lebih efisien ketimbang dengan *brute force*. Adapun pencarian solusi dan pengimplementasian dengan algoritma *divide and conquer* kami secara konsisten menghasilkan hasil perhitungan yang sama dengan algoritma *brute force*. Hal tersebut menandakan bahwa algoritma *divide and conquer* dalam kasus ini selalu menawarkan solusi yang tepat.

Apabila ditelaah lebih lanjut, efisiensi algoritma *divide and conquer* secara eksponensial meningkat dibandingkan dengan *brute force* seiring bertambahnya jumlah titik pada himpunan titik yang diuji. Meskipun efisiensi meningkat seiring bertambahnya n (jumlah titik), algoritma *divide and conquer* menunjukkan penurunan efisiensi (relatif terhadap *brute force*) seiring

bertambahnya dimensi titik pengujian. Jumlah operasi euclidean pada algoritma *brute force* tetap konstan seiring pertambahan dimensi, tetapi pada algoritma *divide and conquer* justru terus bertambah. Bahkan, pada test case 9 dan 10, jumlah operasi euclidean dari algoritma *divide and conquer* secara signifikan melampaui operasi euclidean dari algoritma *brute force*.

Oleh karena itu, kita tidak serta merta dapat mengatakan bahwa “algoritma *divide and conquer* selalu lebih efisien dibandingkan dengan *brute force*”. Pernyataan tersebut merupakan sebuah kekeliruan karena efisiensi dari algoritma *divide and conquer* pada pencarian *closest pairs* di R^n bergantung pada jumlah dimensi perhitungan, sedangkan algoritma *brute force* tidak tergantung pada dimensi. Lain halnya dengan progresi dimensi, progresi n (jumlah titik) pada algoritma *divide and conquer* dalam pencarian *closest pairs* di R^n akan berimbas pada peningkatan efisiensi relatif terhadap algoritma *brute force*.

BAB V

PENUTUP

5.1 Kesimpulan






Algoritma *divide and conquer* merupakan sebuah strategi algoritma yang menawarkan alternatif skema penyelesaian persoalan dengan banyak keuntungan ketimbang algoritma *brute force*. Namun, implementasi *divide and conquer* dalam mencari solusi dari sebuah persoalan harus dilakukan secara tepat guna. Pada pengujian yang dilakukan dalam laporan ini, terbukti bahwa algoritma *divide and conquer* tidak selalu lebih efisien ketimbang algoritma *brute force*. Faktor jumlah atau progresi dimensi memainkan peran dalam kompleksitas algoritma *divide and conquer*. Oleh karena itu, sebuah strategi algoritma selayaknya digunakan dengan tepat guna untuk menghasilkan efisiensi dan efektivitas pengimplementasian yang optimal. Sebuah algoritma tidak serta merta dapat dikatakan sebagai solusi terbaik dalam menyelesaikan suatu persoalan. Terdapat faktor-faktor yang harus ditinjau dalam memilih algoritma yang optimal dalam menyelesaikan sebuah persoalan karena faktor-faktor tersebut bisa saja memengaruhi efisiensi seiring progresi faktor-faktor tersebut. Adapun optimalisasi efisiensi dapat dilakukan untuk mengatasi hal-hal demikian, tetapi tentunya menjadi tantangan tersendiri.

Lampiran

Repository GitHub

https://github.com/melvinkj/Tucil2_13521052_13521094.git

Checkpoint Pengerjaan

No	Poin	Ya	Tidak
1.	Program berhasil dikompilasi tanpa ada kesalahan.		
2.	Program berhasil <i>running</i> .		
3.	Program dapat menerima masukan dan menuliskan luaran.		
4.	Luaran program sudah benar (solusi <i>closest pair</i> benar)		
5.	Bonus 1 dikerjakan		
6.	Bonus 2 dikerjakan	