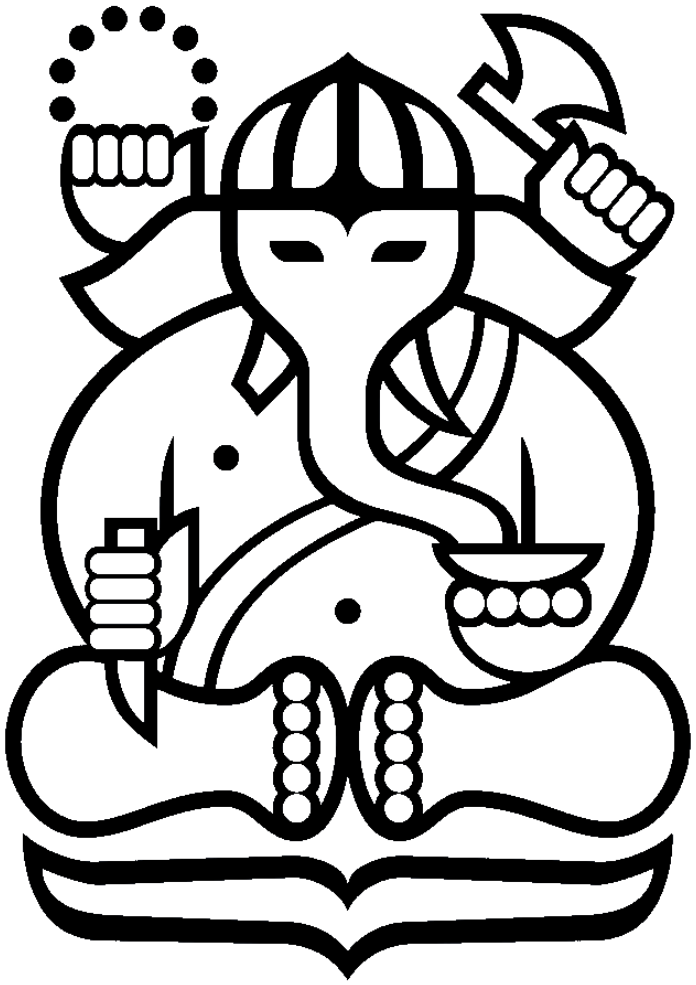
Tugas Kecil 2 IF2211 Strategi Algoritma

Penyelesaian Mencari Pasangan Titik Terdekat dalam ruang 3D dengan Algoritma Divide and Conquer



Disusun oleh :
13521072 - Irsyad Nurwidianto Basuki

INSTITUT TEKNOLOGI BANDUNG
2023

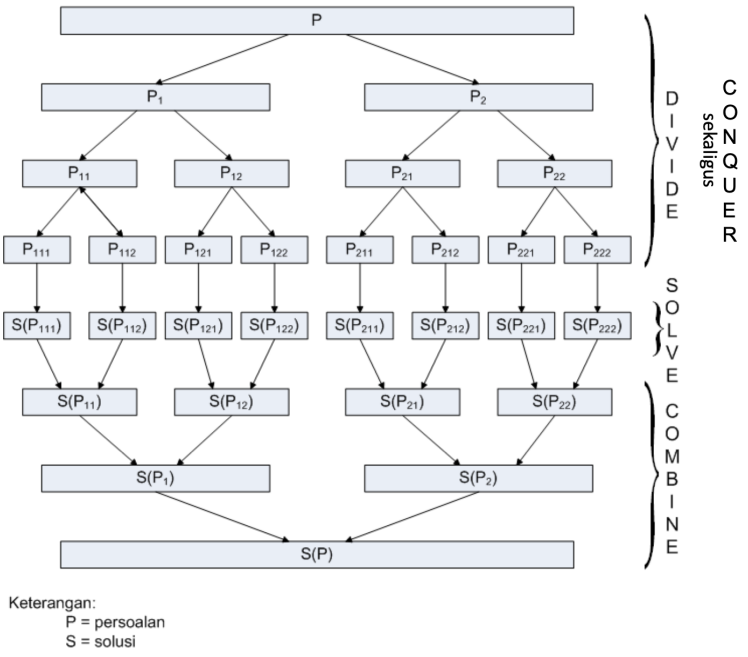
DAFTAR ISI

BAB 1
DESKRIPSI MASALAH DAN ALGORITMA

1.1 Algoritma *Divide and Conquer*

Berdasarkan namanya, *Divide* memiliki arti, membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya berukuran hampir sama), serta *Conquer* berarti menyelesaikan masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar). Di tahap terakhir terdapat tahap *Combine*, yaitu mengabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

Dari penjelasan diatas dapat disimpulkan bahwa algoritma *Divide and Conquer* adalah teknik desain algoritma yang sering digunakan untuk memecahkan masalah kompleks dengan cara membaginya menjadi sub masalah yang lebih kecil, menyelesaikan sub masalah tersebut secara terpisah, dan kemudian menggabungkan solusi dari submasalah-submasalah tersebut untuk mendapatkan solusi untuk masalah awal yang lebih besar. Berikut merupakan ilustrasi dari algoritma *Divide and Conquer* :



Obyek persoalan yang dibagi merupakan masukan atau *instances* persoalan yang berukuran n seperti, tabel (larik), matriks, eksponen, polinom, dan lain - lain bergantung persoalannya. Tiap-tiap upa-persoalan memiliki karakteristik yang sama *(the same type)* dengan karakteristik persoalan semula sehingga metode *Divide and Conquer* lebih natural diungkapkan dalam skema rekursif.

1.2 Permasalahan Mencari Pasangan Titik Terdekat dalam Ruang 3D

Permasalahan pencarian titik terdekat di ruang tiga dimensi merupakan masalah pencarian dua titik terdekat yang tersebar dalam ruang tiga dimensi. Permasalahan ini merupakan generalisasi dari *Closest Pair Problem in 2D*. Untuk menyelesaikan permasalahan pasangan terdekat dalam ruang tiga dimensi, dibutuhkan sebuah algoritma yang dapat mencari dengan efisien seluruh kemungkinan pasangan titik di dalam ruang tiga dimensi dan mengidentifikasi pasangan titik dengan jarak terdekat di antara mereka. Salah satu algoritma yang sering digunakan untuk menyelesaikan permasalahan pasangan terdekat dalam ruang tiga dimensi adalah algoritma Divide and Conquer. Algoritma ini bekerja dengan cara membagi himpunan titik menjadi subset-subset yang lebih kecil dan menyelesaikan masalah secara rekursif untuk setiap subset hingga pasangan titik dengan jarak terdekat berhasil ditemukan. Pencarian titik terdekat dibantu dengan menggunakan rumus Euclidean :

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

1.3 Algoritma Penyelesaian Mencari Pasangan Titik Terdekat dalam Ruang 3D dengan Pendekatan Divide and Conquer

Penulis menggunakan pendekatan *Divide and Conquer* untuk menyelesaikan permasalahan mencari pasangan titik terdekat dalam ruang 3D. Adapun langkah - langkahnya sebagai berikut:

1. Program menerima input dari user berupa pilihan menampung jumlah titik - titik secara manual atau secara acak. Apabila secara manual program akan menerima input dari user yang berupa jumlah titik yang akan disebar.
2. Program akan menyebar titik - titik dengan jumlah yang telah ditentukan.
3. Program akan membaca semua titik dan menyimpan koordinat sumbu x, sumbu y, dan sumbu z dari titik - titik tersebut di sebuah array.
4. Program akan mengurutkan titik - titik tersebut berdasarkan koordinat sumbu x secara membesar.
5. Program akan membagi dua array tersebut menjadi array sisi kiri dan array sisi kanan secara seimbang. Dilakukan secara rekursif
6. Terdapat kasus basis pada fungsi rekursif yang mencari beberapa (dua atau tiga) titik terdekat yang menggunakan algoritma *Brute Force* dan akan digunakan rumus Euclidean untuk mencari jarak titik - titik tersebut..
7. Selanjutnya, program akan melakukan pencarian titik - titik terdekat yang berada di dekat bidang pemisah kedua sisi, atau biasa dinamakan *strip*. Apabila terdapat jarak antar titik yang lebih kecil daripada jarak antar titik yang dicari dari kiri dan kanan sisi maka jarak tersebut akan disimpan

sebagai jarak terdekat. Jika tidak, maka akan menyimpan jarak terdekat yang ditemukan dari pencarian basis (*Brute Force*).

8. Akhirnya, program akan menampilkan jarak terdekat dari titik - titik yang ada serta menampilkan koordinat dari dua titik yang memiliki jarak terdekat. Program juga akan menampilkan koordinat sepasang titik yang jaraknya terdekat serta nilai jaraknya, banyaknya operasi perhitungan rumus Euclidean, waktu eksekusi program dalam detik serta spesifikasikan komputer yang digunakan, dan penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya.
9. Program juga akan melakukan pencarian titik terdekat dalam ruang tiga dimensi dengan menggunakan algoritma *Brute Force*. Hal ini dilakukan untuk membandingkan berapa kali perhitungan rumus Euclidean digunakan yang menyimpulkan keefektifan algoritma yang digunakan.

BAB 2

SOURCE CODE PROGRAM

2.1 Pranala Program

Link repository : *github.com/irsyadnb/Tucil2_13521072*

2.2 Source Code Program

main.py

```
start = True

while(start):
    print("Welcome to Closest Pair Problem using Divide and Conquer Algorithm")
    print("Select how do you wanna inser the points : ")
    print("1. Manual Insertion")
    print("2. Generate Random")
    while True:
        try:
            inputs = int(input("> "))
            break
        except ValueError:
            print("Invalid entry. Please try again")
    while(inputs!=1 and inputs != 2):
        print("Welcome to Closest Pair Problem")
        print("Select how do you wanna inser the points : ")
        print("1. Manual Insertion")
        print("2. Generate Rnadom")
        while True:
            try:
                inputs = int(input("> "))
                break
            except ValueError:
                print("Invalid entry. Please try again")
    start = False
    flag = False
    if (inputs == 1):
        while True:
            try:
                points = int(input("Insert n points you would like to try : "))
                if(points < 2):
                    print("The minimum points is 2!")
                else:
                    flag = True
                    break
            except ValueError:
                print("Invalid entry. Please try again")
        start = time.time()
        vector, ax= pointMakerManual(points)
    else:
        start = time.time()
        print("Generating points...")
        vector, ax = pointMaker()
```

```
pointsSortedByX = quickSort(vector)

closestPair, distance, ctr = divideAndConquer(pointsSortedByX)
end = time.time()

dncTime = end - start

print("Using bruteforce algorithm... (for comparison purposes)")
start = time.time()
test = 999
eucCtrbf = 0
for i in range(len(vector)):
    for j in range(i+1,len(vector)):
        if(eucDist(vector[i], vector[j]) < test):
            test = eucDist(vector[i], vector[j])
            eucCtrbf += 1
            if(test == distance):
                point1 = vector[i]
                point2 = vector[j]
end = time.time()

bfTime = end - start

xpair = [point1[0], point2[0]]
ypair = [point1[1], point2[1]]
zpair = [point1[2], point2[2]]

ax.scatter(xpair,ypair,zpair,c='red',s=80)
print()
print("===Divide & Conquer===")
print("^Nearest Points^")
print("Point 1 <x,y,z> : ", point1)
print("Point 2 <x,y,z> : ", point2)
print("Euclidean Distance  : ", distance)
print(test)
print("Euclidean distance used counter on Divide & Conquer : ", ctr)
print("Euclidean distance used counter on BruteForce : ", eucCtrbf)
print("Execution time using Divide & Conquer : ", dncTime, "seconds")
print("Execution time using BruteForce : ", bfTime, "seconds")
print("Computer Specifitation : ", platform.platform())
```

```
print("Do you want to visualize?")
print("(Y/y) or (N/n)")
v = str(input("> "))
if(v == "Y" or v == "y"):
    plt.show()
print("Goodbye!")
start = False
```

points.py

```python
def pointMaker(): #Random Point Generator
    x = np.random.randint(500)
    fig = plt.figure(figsize=(5,5))

    ax = fig.add_subplot(111, projection='3d')

    xs = np.random.random(x)
    ys = np.random.random(x)
    zs = np.random.random(x)
    scatter = ax.scatter(xs,ys,zs)
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")

    points = np.array(scatter._offsets3d)

    vector = []
    for i in range(len(points[0])):
        list = []
        list.append(points[0][i])
        list.append(points[1][i])
        list.append(points[2][i])
        vector.append(list)

    return vector, ax
```

```python
def pointMakerManual(n): #Manual Point Generator
    fig = plt.figure(figsize=(5,5))

    ax = fig.add_subplot(111, projection='3d')

    xs = np.random.random(n)
    ys = np.random.random(n)
    zs = np.random.random(n)
    scatter = ax.scatter(xs,ys,zs)
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")

    points = np.array(scatter._offsets3d)

    vector = []
    for i in range(len(points[0])):
        list = []
        list.append(points[0][i])
        list.append(points[1][i])
        list.append(points[2][i])
        vector.append(list)

    return vector, ax
```

utility.py

```
eucCtr = 0

def quickSort(points): # QuickSort Divide & Conquer
    size = len(points)
    mid = size // 2

    if(size <= 1):
        return points
    else:
        pivot = points[mid]
        lessThanPivot = []
        biggerThanPivot = []
        isPivot = []
        for i in range(len(points)):
            if points[i] < pivot:
                lessThanPivot.append(points[i])
            elif points[i] == pivot:
                isPivot.append(points[i])
            else:
                biggerThanPivot.append(points[i])

    return quickSort(lessThanPivot) + isPivot + quickSort(biggerThanPivot)

def eucDist(point1, point2): # Basic euclidean distance
    return np.sqrt( pow(point1[0] - point2[0], 2) + pow(point1[1] - point2[1], 2) + pow(point1[2] - point2[2], 2) )
```

```
def eucDistFewPoints(point, size): #Solving few close pair points using brute force
    points = []
    temp = 9999
    for i in range(size):
        for j in range(i+1, size):
            if(eucDist(point[i], point[j]) < temp):
                temp = eucDist(point[i], point[j])
                points.append(point[i])
                points.append(point[j])
                global eucCtr
                eucCtr += 1
    return points, temp, eucCtr
```

divideNconquer.py

```python
def divideAndConquer(points):
    size = len(points)

    # - Base Case
    if(size<4):
        closestpoints, closestDist, ctr = eucDistFewPoints(points, size)
        return closestpoints, closestDist, ctr

    # - Divide
    mid = size // 2  # Dividing the array into two sides
    leftside = points[:mid]
    rightside = points[mid:]

    # - Recursively conquering left and right
    leftpoints, deltaL, ctr = divideAndConquer(leftside)
    rightpoints, deltaR, ctr = divideAndConquer(rightside)

    closestDist = min(deltaL, deltaR)

    newClosestBetween = False
```

```python
    # Finding distance near middle strip and comparing with earlier closest distance
    for i in range(len(leftside)):
        for j in range(len(rightside)):
            # Distance between left side and right side
            distanceNearMid = abs(leftside[i][0] - rightside[j][0])

            # Distance near middle must be smaller and check also for its euclidean distance
            if(distanceNearMid < closestDist and eucDist(leftside[i], rightside[j]) < closestDist):
                closestpoints = []
                closestDist = eucDist(leftside[i], rightside[j])
                closestpoints.append(leftside[i])
                closestpoints.append(rightside[j])
                ctr+=2
                newClosestBetween = True
```

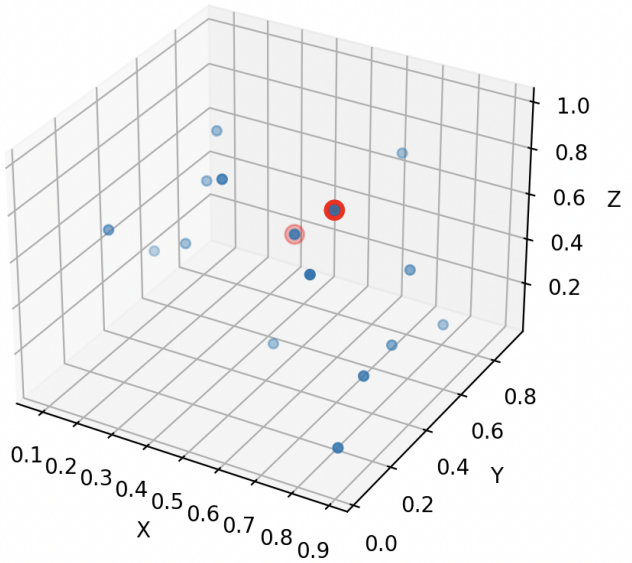
```
    # If no pair of points near middle strip with distance less than earlier closest distance
    # Keep using the earlier closes distance
    if not newClosestBetween :
        # Assign two points with the earlier closest distance
        closestpoints = []
        if(deltaL < deltaR):
            closestpoints.append(leftpoints)
        else:
            closestpoints.append(rightpoints)

    return closestpoints, closestDist, ctr
```

BAB 3

MASUKAN DAN LUARAN PROGRAM

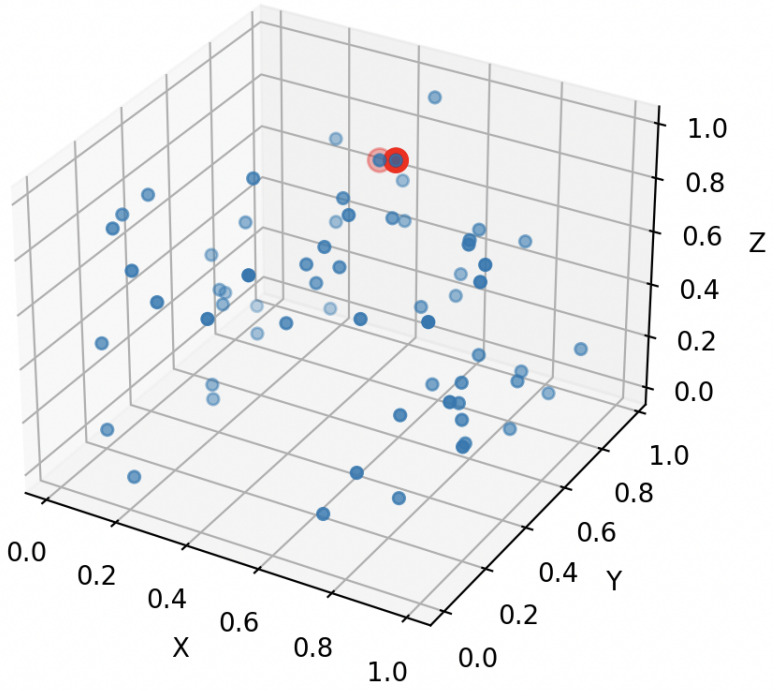
N = 16

```
Welcome to Closest Pair Problem using Divide and Conquer Algorithm
Select how do you wanna inser the points :
1. Manual Insertion
2. Generate Random
> 1
Insert n points you would like to try : 16
Using bruteforce algorithm... (for comparison purposes)

===Divide & Conquer===
^Nearest Points^
Point 1 <x,y,z> :  [0.6011144455764604, 0.5936656484237713, 0.4433052279694164]
Point 2 <x,y,z> :  [0.6855144583718424, 0.5345983714990957, 0.37391154768927337]
Euclidean Distance  :  0.12420864795166106
0.12420864795166106
Euclidean distance used counter on Divide & Conquer :  8
Euclidean distance used counter on BruteForce :  120
Execution time using Divide & Conquer :  0.16876006126403809 seconds
Execution time using BruteForce :  0.0002658367156982422 seconds
Computer Specifitation :  macOS-13.2-x86_64-i386-64bit
Do you want to visualize?
(Y/y) or (N/n)
> n
Goodbye!
```



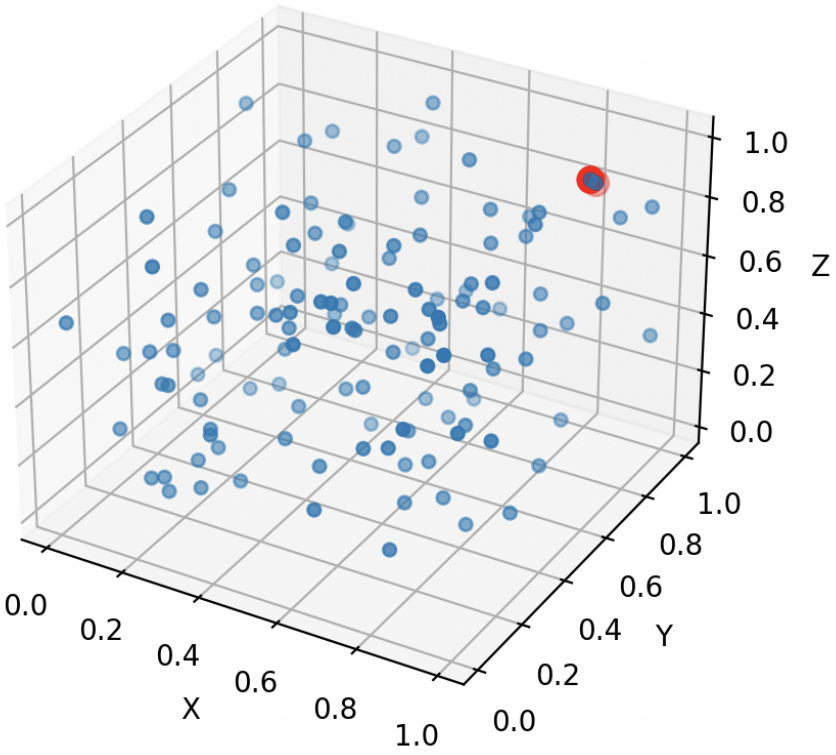
N = 64

```
Welcome to Closest Pair Problem using Divide and Conquer Algorithm
Select how do you wanna inser the points :
1. Manual Insertion
2. Generate Random
> 1
Insert n points you would like to try : 64
Using bruteforce algorithm... (for comparison purposes)

===Divide & Conquer===
^Nearest Points^
Point 1 <x,y,z> :  [0.2551964176195455, 0.3141854769946414, 0.5691653977167196]
Point 2 <x,y,z> :  [0.27753859166229045, 0.27946896319297454, 0.5976849364027326]
Euclidean Distance  :  0.05017741681633903
0.05017741681633903
Euclidean distance used counter on Divide & Conquer :  36
Euclidean distance used counter on BruteForce :  2016
Execution time using Divide & Conquer :  0.2028789520263672 seconds
Execution time using BruteForce :  0.003988027572631836 seconds
Computer Specifitation :  macOS-13.2-x86_64-i386-64bit
Do you want to visualize?
(Y/y) or (N/n)
```



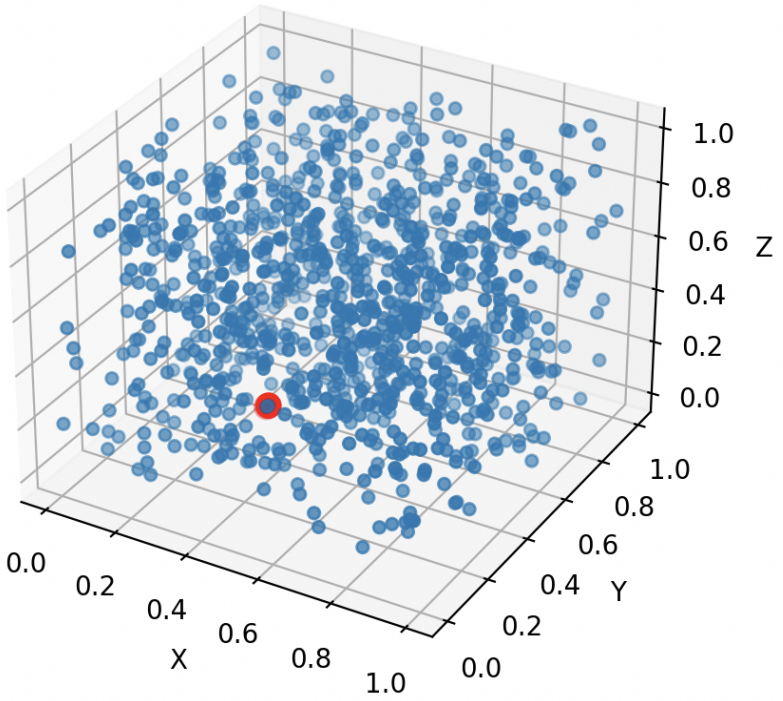
N = 128

```
Welcome to Closest Pair Problem using Divide and Conquer Algorithm
Select how do you wanna inser the points :
1. Manual Insertion
2. Generate Random
> 1
Insert n points you would like to try : 128
Using bruteforce algorithm... (for comparison purposes)

===Divide & Conquer===
^Nearest Points^
Point 1 <x,y,z> :  [0.896193386988523, 0.21326975355425004, 0.6675688316003558]
Point 2 <x,y,z> :  [0.9056934274015036, 0.22630104778120352, 0.6733681732519219]
Euclidean Distance  :  0.017137612455335897
0.017137612455335897
Euclidean distance used counter on Divide & Conquer :  74
Euclidean distance used counter on BruteForce :  8128
Execution time using Divide & Conquer :  0.18094110488891602 seconds
Execution time using BruteForce :  0.016656875610351562 seconds
Computer Specifitation :  macOS-13.2-x86_64-i386-64bit
Do you want to visualize?
(Y/y) or (N/n)
```



N = 1000

```
Welcome to Closest Pair Problem using Divide and Conquer Algorithm
Select how do you wanna inser the points :
1. Manual Insertion
2. Generate Random
> 1
Insert n points you would like to try : 1000
Using bruteforce algorithm... (for comparison purposes)

===Divide & Conquer===
^Nearest Points^
Point 1 <x,y,z> :  [0.36287889361418457, 0.3778417905509731, 0.15264512729988222]
Point 2 <x,y,z> :  [0.3592812766026614, 0.374227302769533, 0.14857153542743962]
Euclidean Distance  :  0.0065269840529153566
0.0065269840529153566
Euclidean distance used counter on Divide & Conquer :  514
Euclidean distance used counter on BruteForce :  499500
Execution time using Divide & Conquer :  0.31412792205810547 seconds
Execution time using BruteForce :  0.9687609672546387 seconds
Computer Specifitation :  macOS-13.2-x86_64-i386-64bit
Do you want to visualize?
(Y/y) or (N/n)
```



LAMPIRAN

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil running	✓	
3. Program dapat membaca input / generate sendiri dan memberikan luaran	✓	
4. Luaran program sudah benar (solusi closest pair benar)	✓	
5. Bonus 1 dikerjakan	✓	
6. Bonsu 2 dikerjakan		✓