LAPORAN TUGAS KECIL 2

IF-2211 STRATEGI ALGORITMA

MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA *DIVIDE AND CONQUER*

Disusun oleh:

Tobias Natalio Sianipar 13521090

PROGRAM STUDI TEKNIK INFORMATIKA

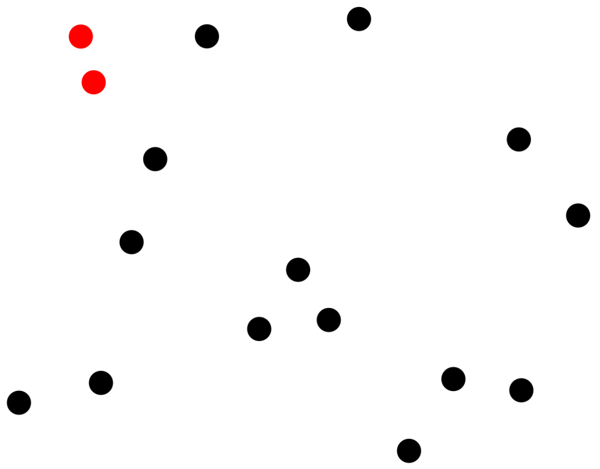
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

BAB I
DESKRIPSI MASALAH

Mahasiswa diminta untuk membuat program yang mencari titik terdekat pada suatu himpunan titik.



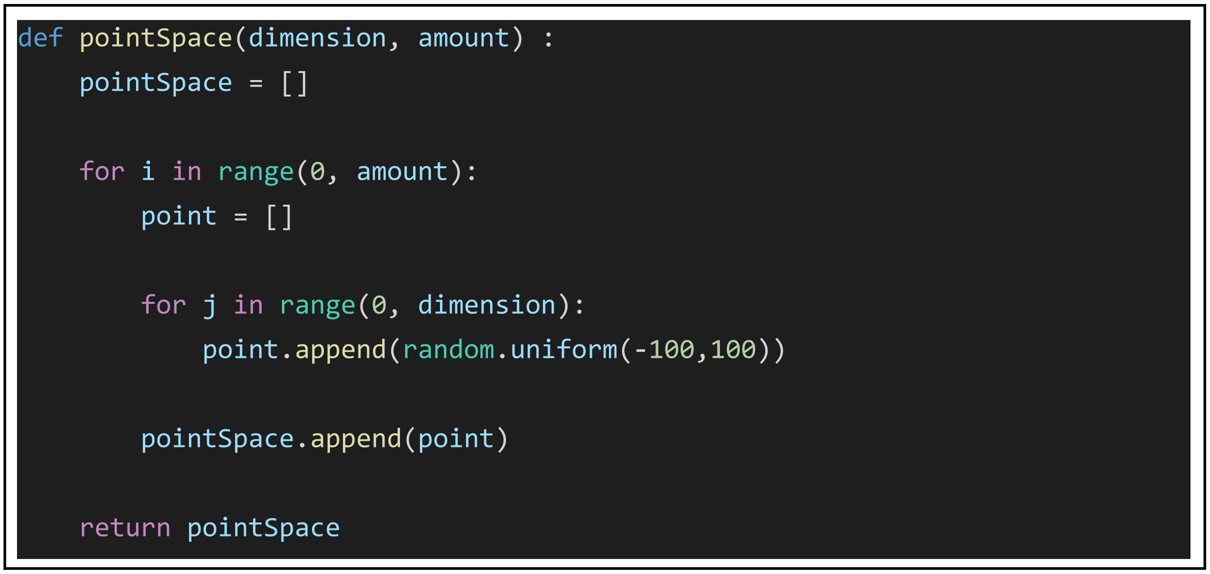
Pencarian titik terdekat ini dapat dilakukan menggunakan *brute force* dan juga *divide and conquer*. Pada tucil ini mahasiswa perlu mengembangkan algoritma *divide and conquer* yang telah diajarkan pada kuliah strategi algortitma, sehingga dapat mencari titik terdekat pada bidang berdimensi lebih dari 2.

BAB II

ALGORITMA

1. Input

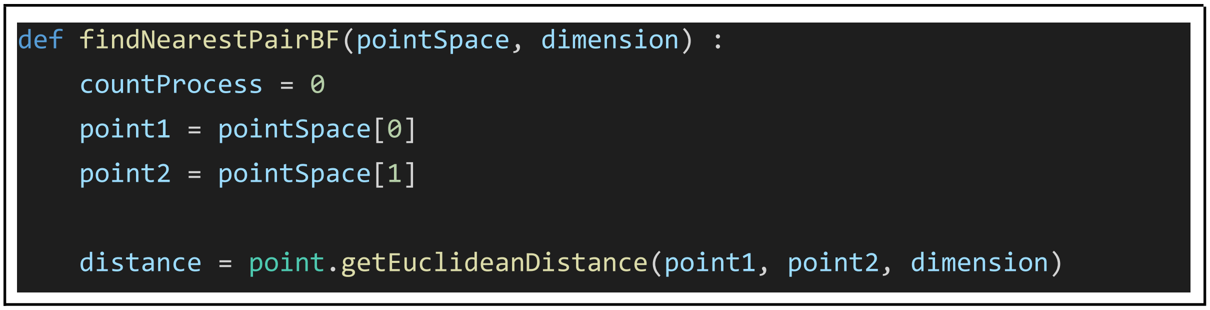
   Pada awal berjalannya program, program akan meminta user masukkan input berupa jumlah dimensi dan jumlah titik, setelah itu program akan menginisiasi titik-titik secara acak dengan menggunakan fungsi **pointSpace** yang menghasilkan array berisi koordinat titik.



```
def pointSpace(dimension, amount) :
    pointSpace = []

    for i in range(0, amount):
        point = []

        for j in range(0, dimension):
            point.append(random.uniform(-100,100))

        pointSpace.append(point)

    return pointSpace
```

2. Solusi Brute Force

   Solusi *brute force* dilakukan dengan cara menghitung jarak tiap titik dengan tiap titik lain menggunakan 2 loop sehingga cara ini berkompleksitas O(n^2)..

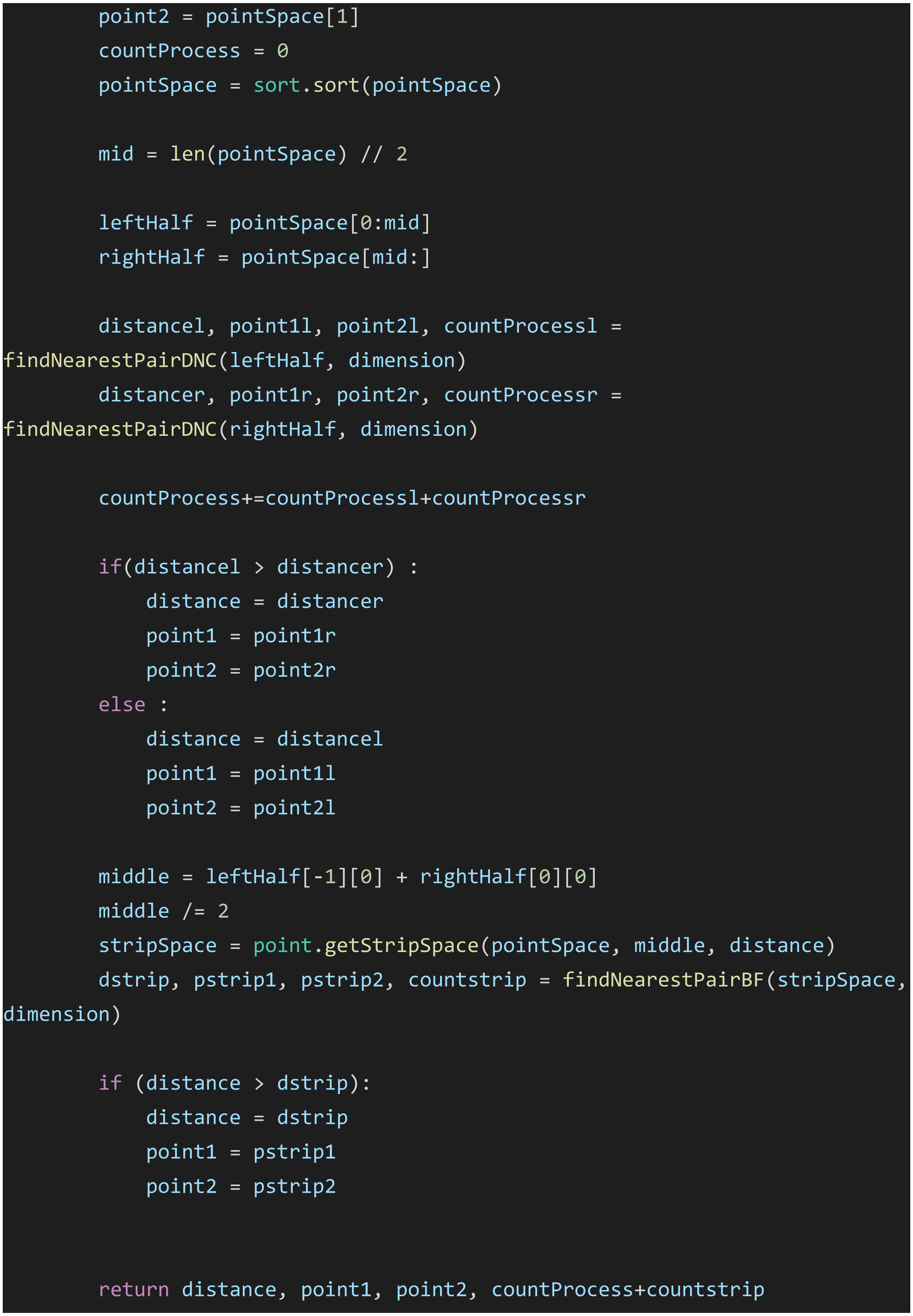


```
def findNearestPairBF(pointSpace, dimension) :
    countProcess = 0
    point1 = pointSpace[0]
    point2 = pointSpace[1]

    distance = point.getEuclideanDistance(point1, point2, dimension)
    for i in range(0, len(pointSpace)) :
        for j in range(i+1, len(pointSpace)) :
            newDistance = point.getEuclideanDistance(pointSpace[i],
pointSpace[j], dimension)
            if (newDistance < distance) :
                point1 = pointSpace[i]
                point2 = pointSpace[j]

                distance = newDistance

            countProcess+=1


    return distance, point1, point2, countProcess
```

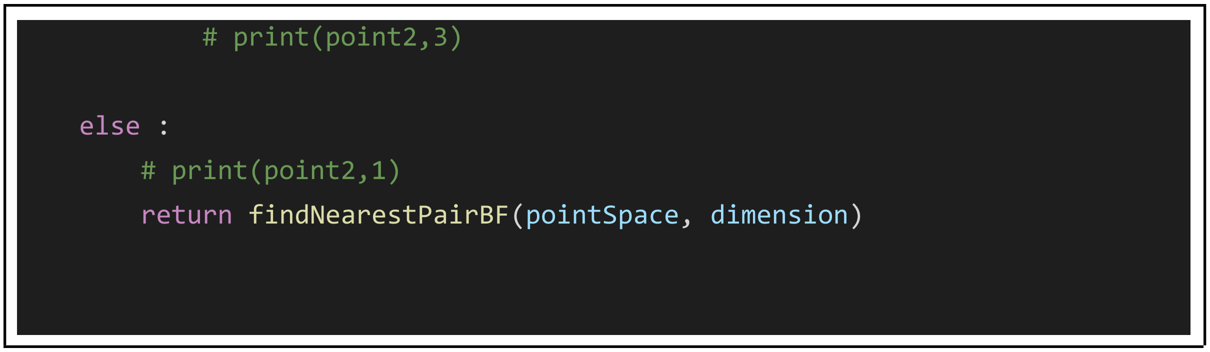
3. Solusi Divide and Conquer

Solusi ini bekerja dengan cara membagi himpunan titik menjadi dua bagian left and right kemudian untuk setiap left dan right akan direkursif sampai array titik bersisa 3 atau kurang, pada array berisi 3 titik tersebut dicari jarak terdekat antara tiga titik tersebut serta dibandingkan dengan jarak nilai ekstrim left dan right, setelah itu jika jarak terkecil merupakan jarak titik ekstrim, maka akan dicari jarak titik terdekat pada area titik ekstrim dan jarak tersebut merupakan jarak terdekat pada di daerah left union right. Kompleksitas algoritma ini dapat dinyatakan sebagai T(n) = 2T(n/2) + cn untuk n > 2 dan T(n) = a untuk n = 2. Sesuai relasi rekurens Teorema Master, T(n) = aT(n/b) + cn^d di mana a = 2, b = 2, d = 1, kompleksitas algoritmanya adalah O(nlog(n)), lebih baik dari solusi Brute Force O(n^2).

```
def findNearestPairDNC(pointSpace, dimension) :


    if(len(pointSpace) > 3) :

        distance = 0
        point1 = pointSpace[0]
```



```
        point2 = pointSpace[1]
        countProcess = 0
        pointSpace = sort.sort(pointSpace)

        mid = len(pointSpace) // 2

        leftHalf = pointSpace[0:mid]
        rightHalf = pointSpace[mid:]

        distancel, point1l, point2l, countProcessl = 
findNearestPairDNC(leftHalf, dimension)
        distancer, point1r, point2r, countProcessr = 
findNearestPairDNC(rightHalf, dimension)

        countProcess+=countProcessl+countProcessr

        if(distancel > distancer) :
            distance = distancer
            point1 = point1r
            point2 = point2r
        else :
            distance = distancel
            point1 = point1l
            point2 = point2l

        middle = leftHalf[-1][0] + rightHalf[0][0]
        middle /= 2
        stripSpace = point.getStripSpace(pointSpace, middle, distance)
        dstrip, pstrip1, pstrip2, countstrip = findNearestPairBF(stripSpace, 
dimension)

        if (distance > dstrip):
            distance = dstrip
            point1 = pstrip1
            point2 = pstrip2


        return distance, point1, point2, countProcess+countstrip
```

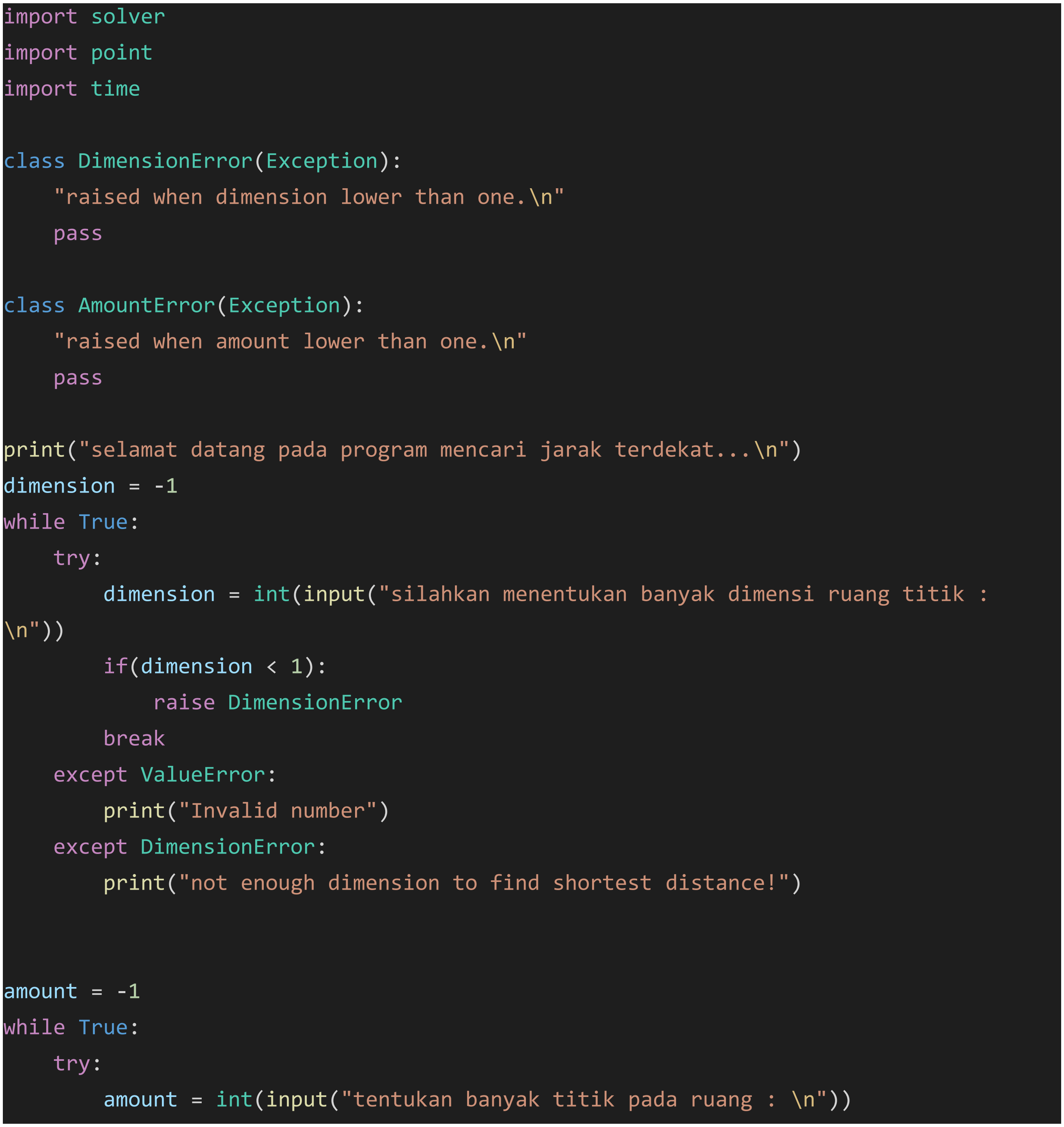


```
            # print(point2,3)

    else :
        # print(point2,1)
        return findNearestPairBF(pointSpace, dimension)
```

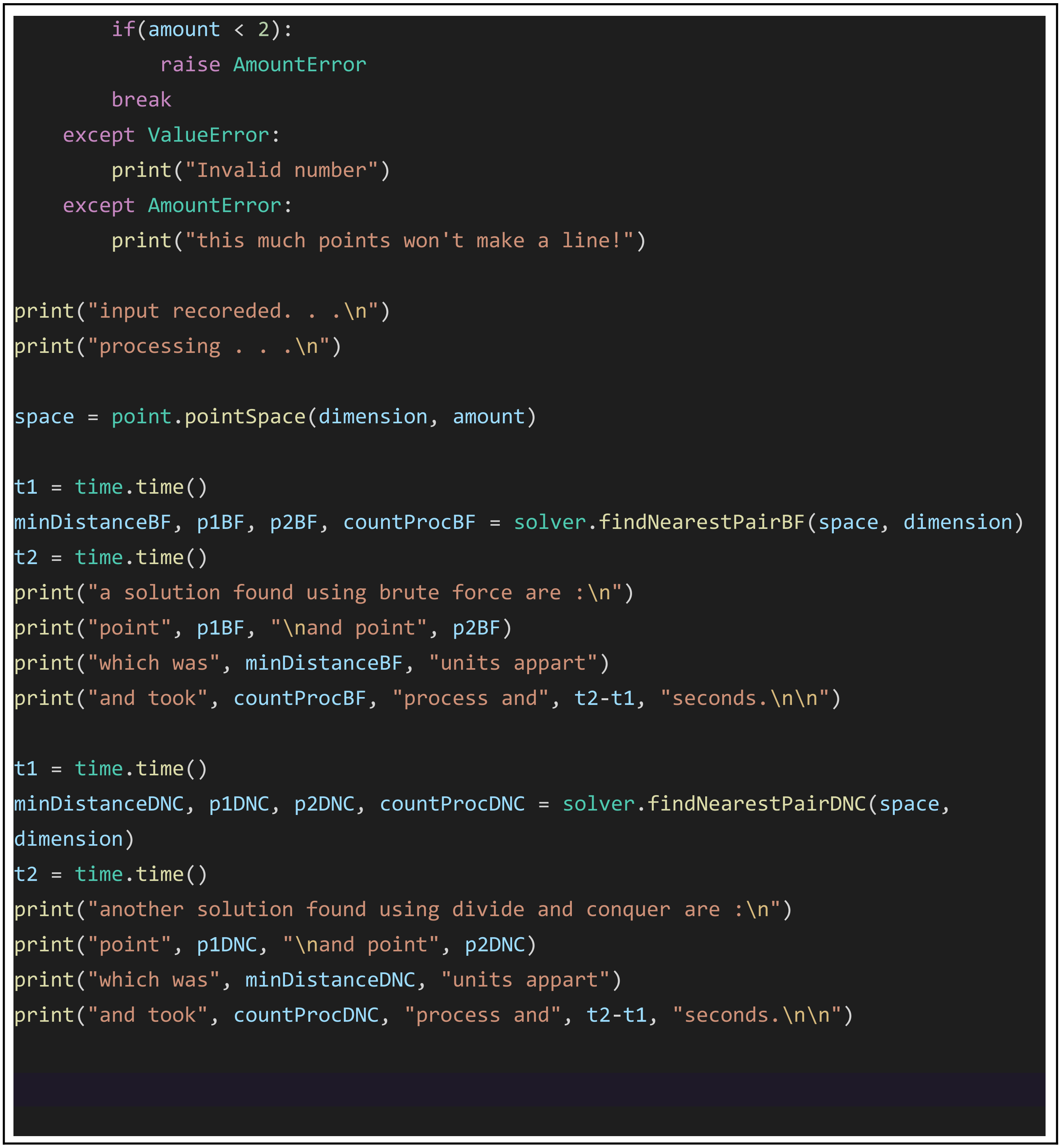
BAB III

SOURCE CODE

Source code dari file **main.py**

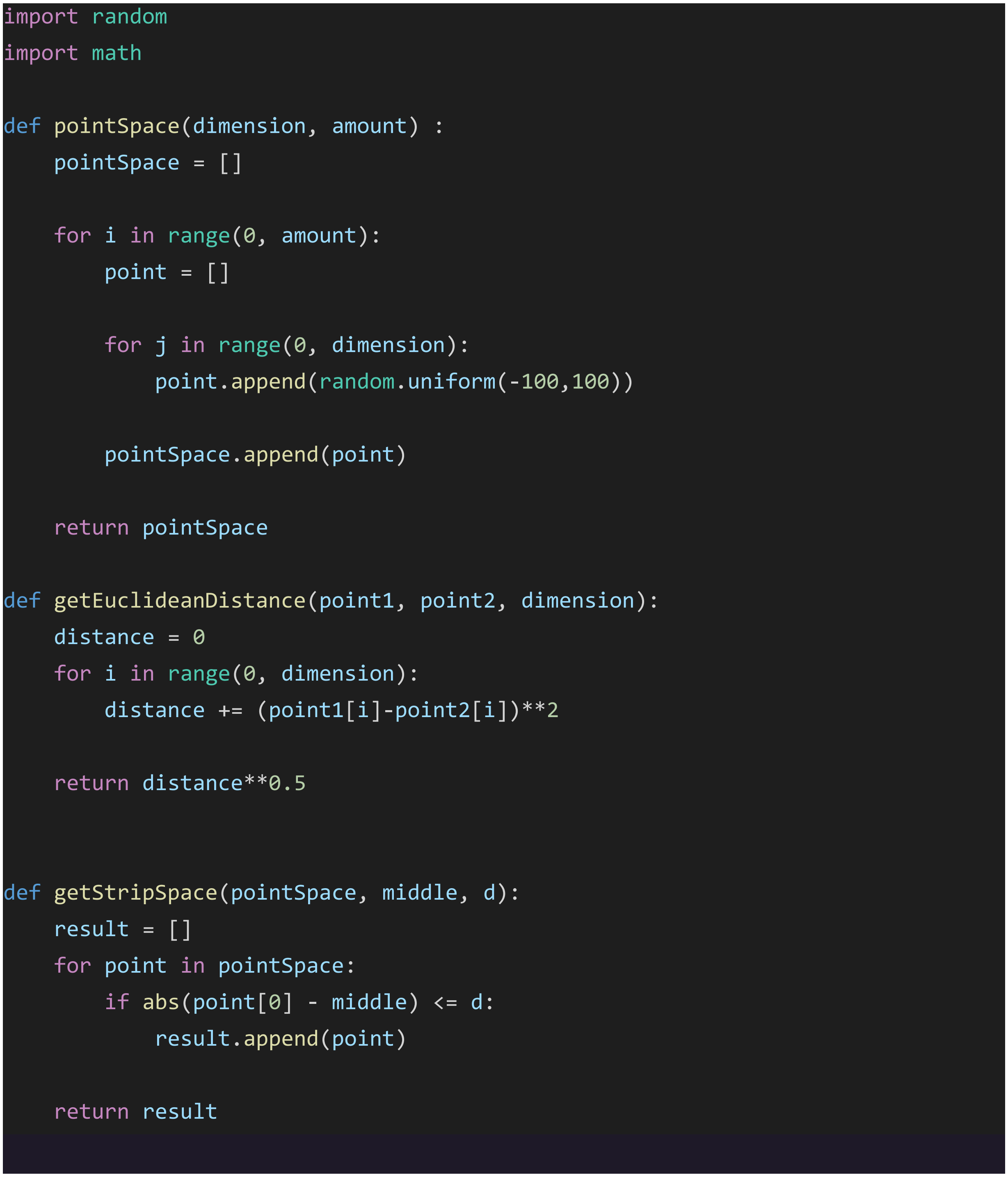


```
import solver
import point
import time

class DimensionError(Exception):
    "raised when dimension lower than one.\n"
    pass

class AmountError(Exception):
    "raised when amount lower than one.\n"
    pass

print("selamat datang pada program mencari jarak terdekat...\n")
dimension = -1
while True:
    try:
        dimension = int(input("silahkan menentukan banyak dimensi ruang titik : 
\n"))
        if(dimension < 1):
            raise DimensionError
        break
    except ValueError:
        print("Invalid number")
    except DimensionError:
        print("not enough dimension to find shortest distance!")


amount = -1
while True:
    try:
        amount = int(input("tentukan banyak titik pada ruang : \n"))
```



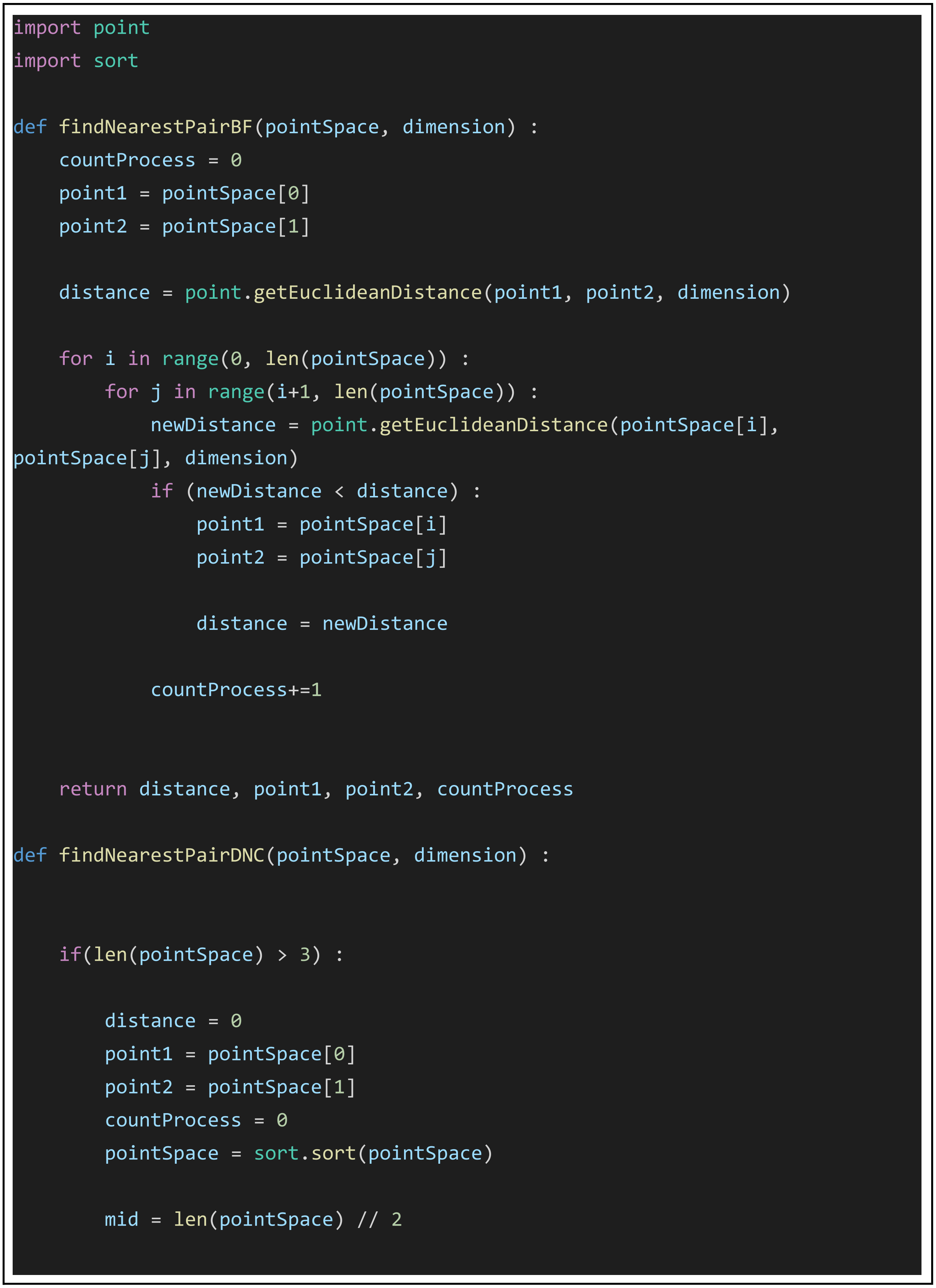
```
        if(amount < 2):
            raise AmountError
        break
    except ValueError:
        print("Invalid number")
    except AmountError:
        print("this much points won't make a line!")

print("input recoreded. . .\n")
print("processing . . .\n")

space = point.pointSpace(dimension, amount)

t1 = time.time()
minDistanceBF, p1BF, p2BF, countProcBF = solver.findNearestPairBF(space, dimension)
t2 = time.time()
print("a solution found using brute force are :\n")
print("point", p1BF, "\nand point", p2BF)
print("which was", minDistanceBF, "units appart")
print("and took", countProcBF, "process and", t2-t1, "seconds.\n\n")

t1 = time.time()
minDistanceDNC, p1DNC, p2DNC, countProcDNC = solver.findNearestPairDNC(space,
dimension)
t2 = time.time()
print("another solution found using divide and conquer are :\n")
print("point", p1DNC, "\nand point", p2DNC)
print("which was", minDistanceDNC, "units appart")
print("and took", countProcDNC, "process and", t2-t1, "seconds.\n\n")
```

Source code dari **point.py**

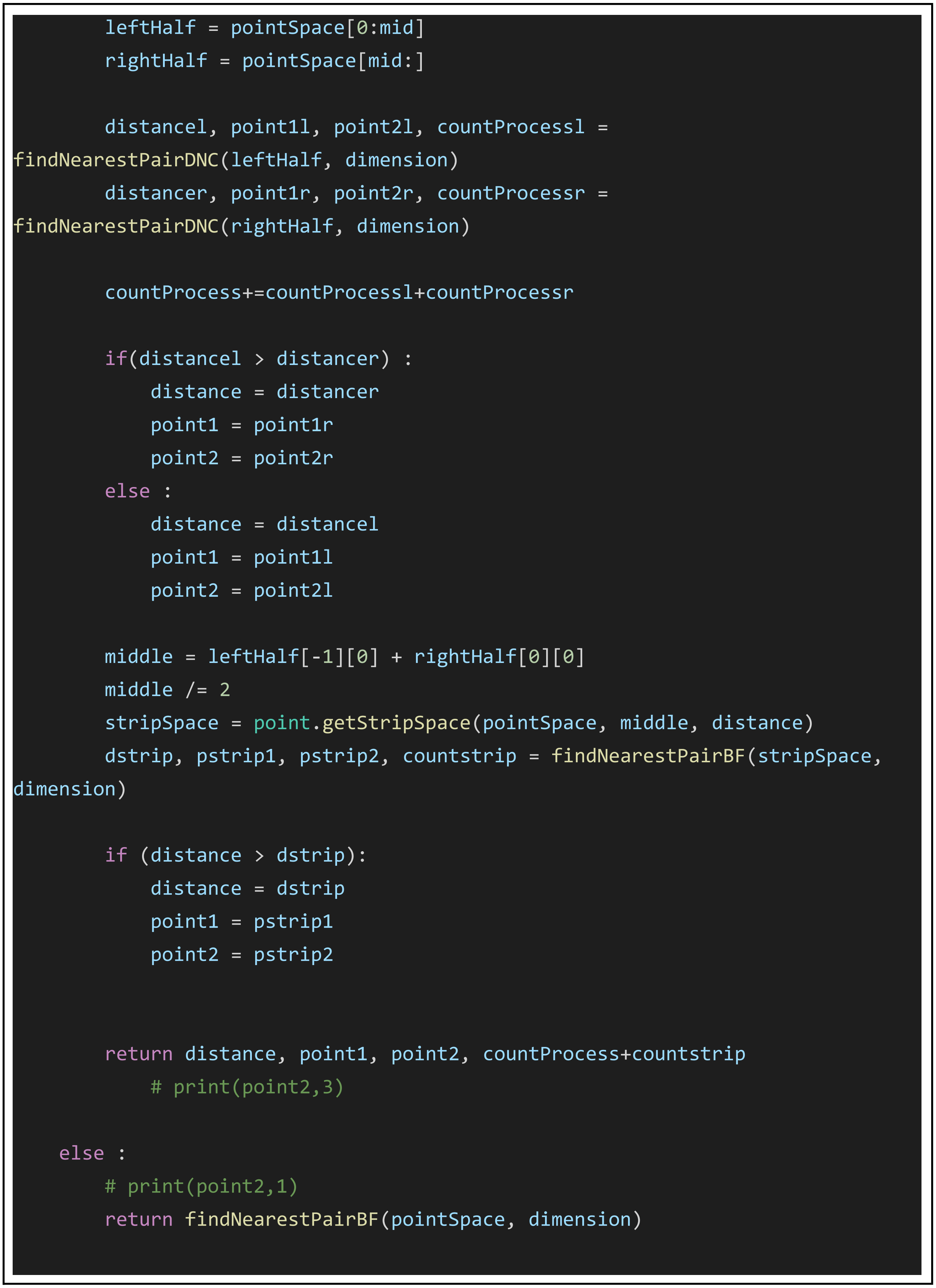


```python
import random
import math

def pointSpace(dimension, amount) :
    pointSpace = []

    for i in range(0, amount):
        point = []

        for j in range(0, dimension):
            point.append(random.uniform(-100,100))

        pointSpace.append(point)

    return pointSpace

def getEuclideanDistance(point1, point2, dimension):
    distance = 0
    for i in range(0, dimension):
        distance += (point1[i]-point2[i])**2

    return distance**0.5


def getStripSpace(pointSpace, middle, d):
    result = []
    for point in pointSpace:
        if abs(point[0] - middle) <= d:
            result.append(point)

    return result
```

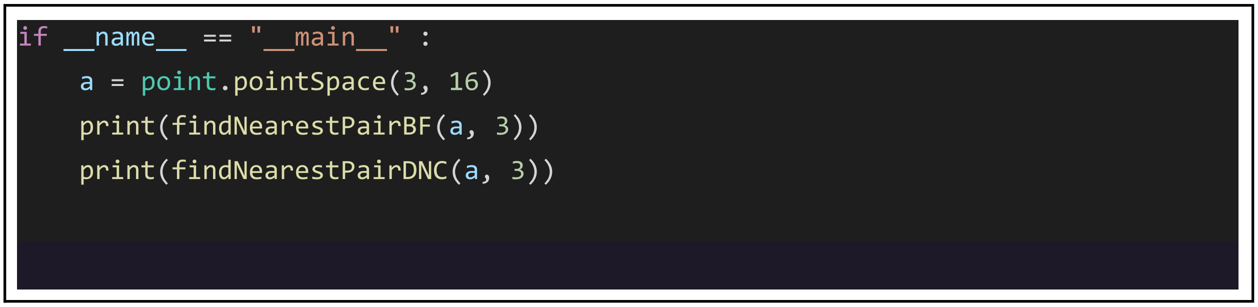
Source code dari **solver.py**



```
import point
import sort

def findNearestPairBF(pointSpace, dimension) :
    countProcess = 0
    point1 = pointSpace[0]
    point2 = pointSpace[1]

    distance = point.getEuclideanDistance(point1, point2, dimension)

    for i in range(0, len(pointSpace)) :
        for j in range(i+1, len(pointSpace)) :
            newDistance = point.getEuclideanDistance(pointSpace[i],
pointSpace[j], dimension)
            if (newDistance < distance) :
                point1 = pointSpace[i]
                point2 = pointSpace[j]

                distance = newDistance

            countProcess+=1


    return distance, point1, point2, countProcess

def findNearestPairDNC(pointSpace, dimension) :


    if(len(pointSpace) > 3) :

        distance = 0
        point1 = pointSpace[0]
        point2 = pointSpace[1]
        countProcess = 0
        pointSpace = sort.sort(pointSpace)

        mid = len(pointSpace) // 2
```

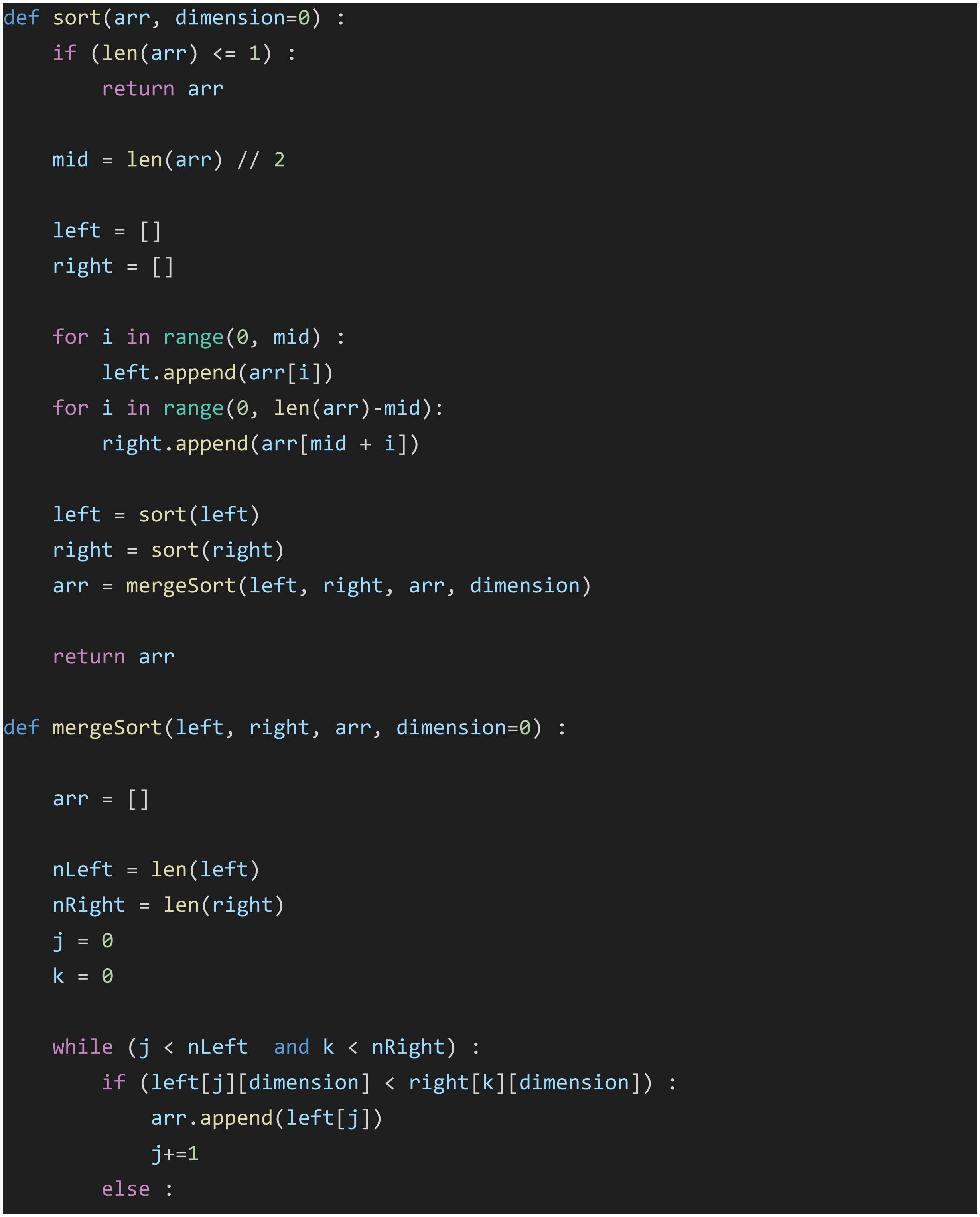


```
        leftHalf = pointSpace[0:mid]
        rightHalf = pointSpace[mid:]

        distancel, point1l, point2l, countProcessl =
findNearestPairDNC(leftHalf, dimension)
        distancer, point1r, point2r, countProcessr =
findNearestPairDNC(rightHalf, dimension)

        countProcess+=countProcessl+countProcessr

        if(distancel > distancer) :
            distance = distancer
            point1 = point1r
            point2 = point2r
        else :
            distance = distancel
            point1 = point1l
            point2 = point2l

        middle = leftHalf[-1][0] + rightHalf[0][0]
        middle /= 2
        stripSpace = point.getStripSpace(pointSpace, middle, distance)
        dstrip, pstrip1, pstrip2, countstrip = findNearestPairBF(stripSpace,
dimension)

        if (distance > dstrip):
            distance = dstrip
            point1 = pstrip1
            point2 = pstrip2


        return distance, point1, point2, countProcess+countstrip
            # print(point2,3)

    else :
        # print(point2,1)
        return findNearestPairBF(pointSpace, dimension)
```

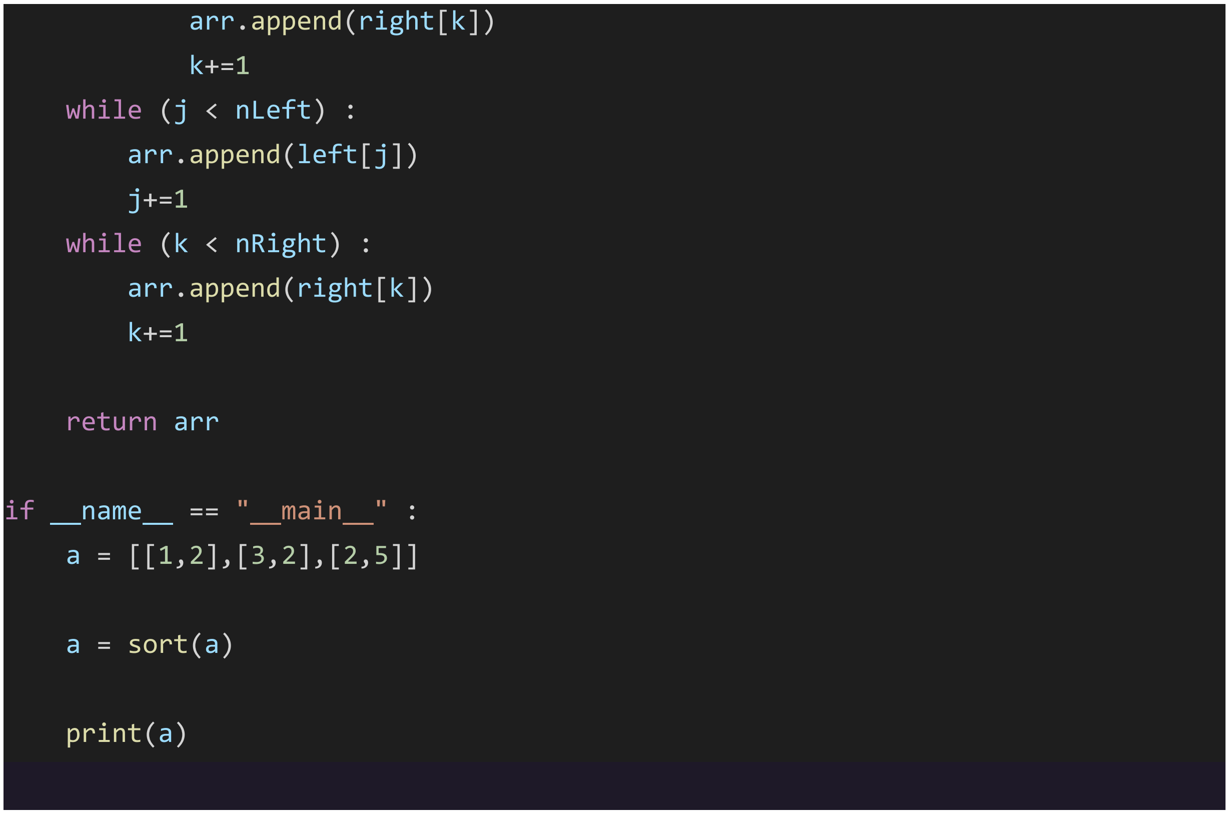


```python
if __name__ == "__main__" :
    a = point.pointSpace(3, 16)
    print(findNearestPairBF(a, 3))
    print(findNearestPairDNC(a, 3))
```

Source code dari **sort.py**



```python
def sort(arr, dimension=0) :
    if (len(arr) <= 1) :
        return arr

    mid = len(arr) // 2

    left = []
    right = []

    for i in range(0, mid) :
        left.append(arr[i])
    for i in range(0, len(arr)-mid):
        right.append(arr[mid + i])

    left = sort(left)
    right = sort(right)
    arr = mergeSort(left, right, arr, dimension)

    return arr

def mergeSort(left, right, arr, dimension=0) :

    arr = []

    nLeft = len(left)
    nRight = len(right)
    j = 0
    k = 0

    while (j < nLeft  and k < nRight) :
        if (left[j][dimension] < right[k][dimension]) :
            arr.append(left[j])
            j+=1
        else :
```

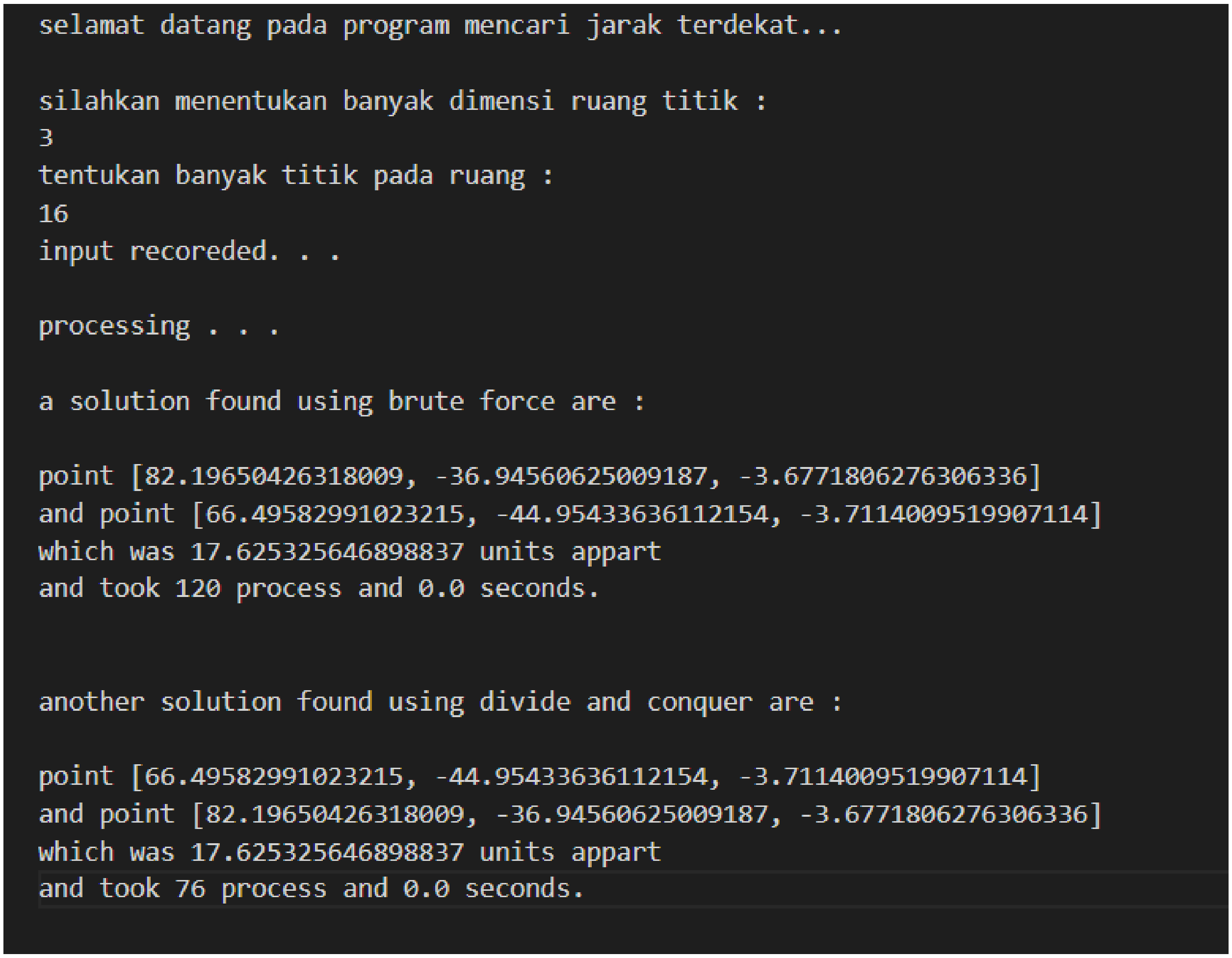


```
            arr.append(right[k])
            k+=1
    while (j < nLeft) :
        arr.append(left[j])
        j+=1
    while (k < nRight) :
        arr.append(right[k])
        k+=1

    return arr

if __name__ == "__main__" :
    a = [[1,2],[3,2],[2,5]]

    a = sort(a)

    print(a)
```

BAB IV

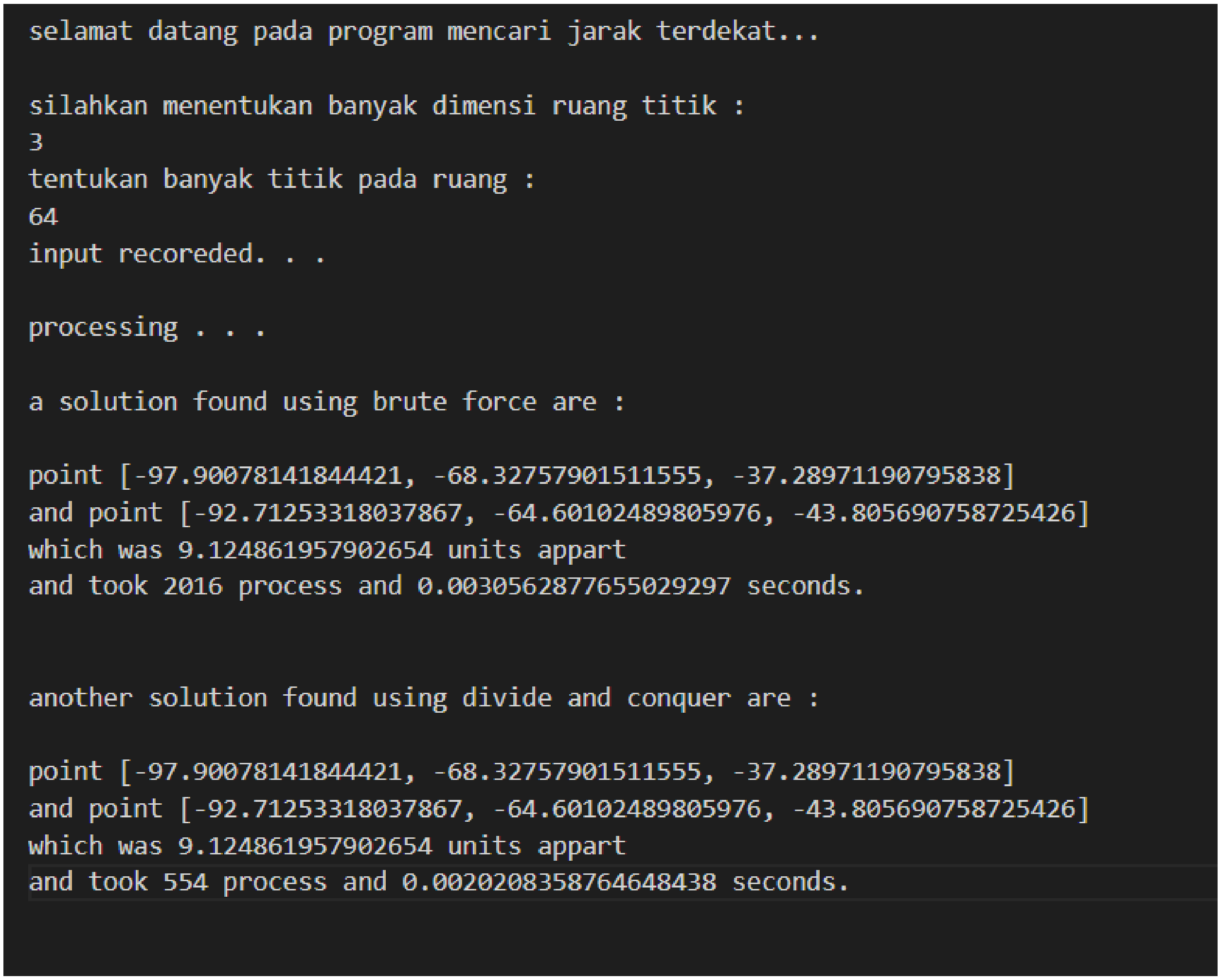
TEST PROGRAM

1. Testcase 1 (n=16)



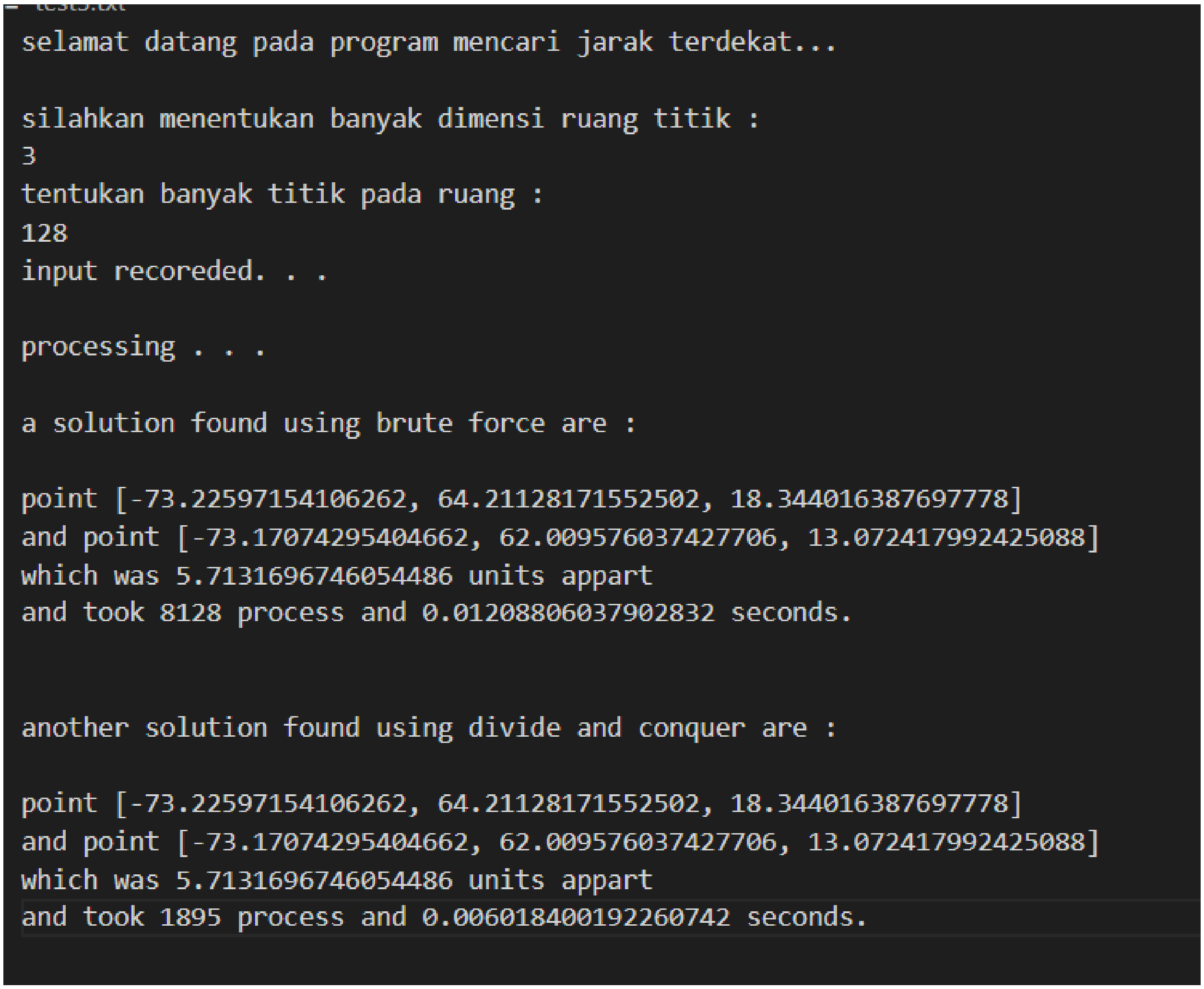
```
selamat datang pada program mencari jarak terdekat...

silahkan menentukan banyak dimensi ruang titik :
3
tentukan banyak titik pada ruang :
16
input recoreded. . .

processing . . .

a solution found using brute force are :

point [82.19650426318009, -36.94560625009187, -3.6771806276306336]
and point [66.49582991023215, -44.95433636112154, -3.7114009519907114]
which was 17.625325646898837 units appart
and took 120 process and 0.0 seconds.


another solution found using divide and conquer are :

point [66.49582991023215, -44.95433636112154, -3.7114009519907114]
and point [82.19650426318009, -36.94560625009187, -3.6771806276306336]
which was 17.625325646898837 units appart
and took 76 process and 0.0 seconds.
```

2. Testcase 2 (n=64)



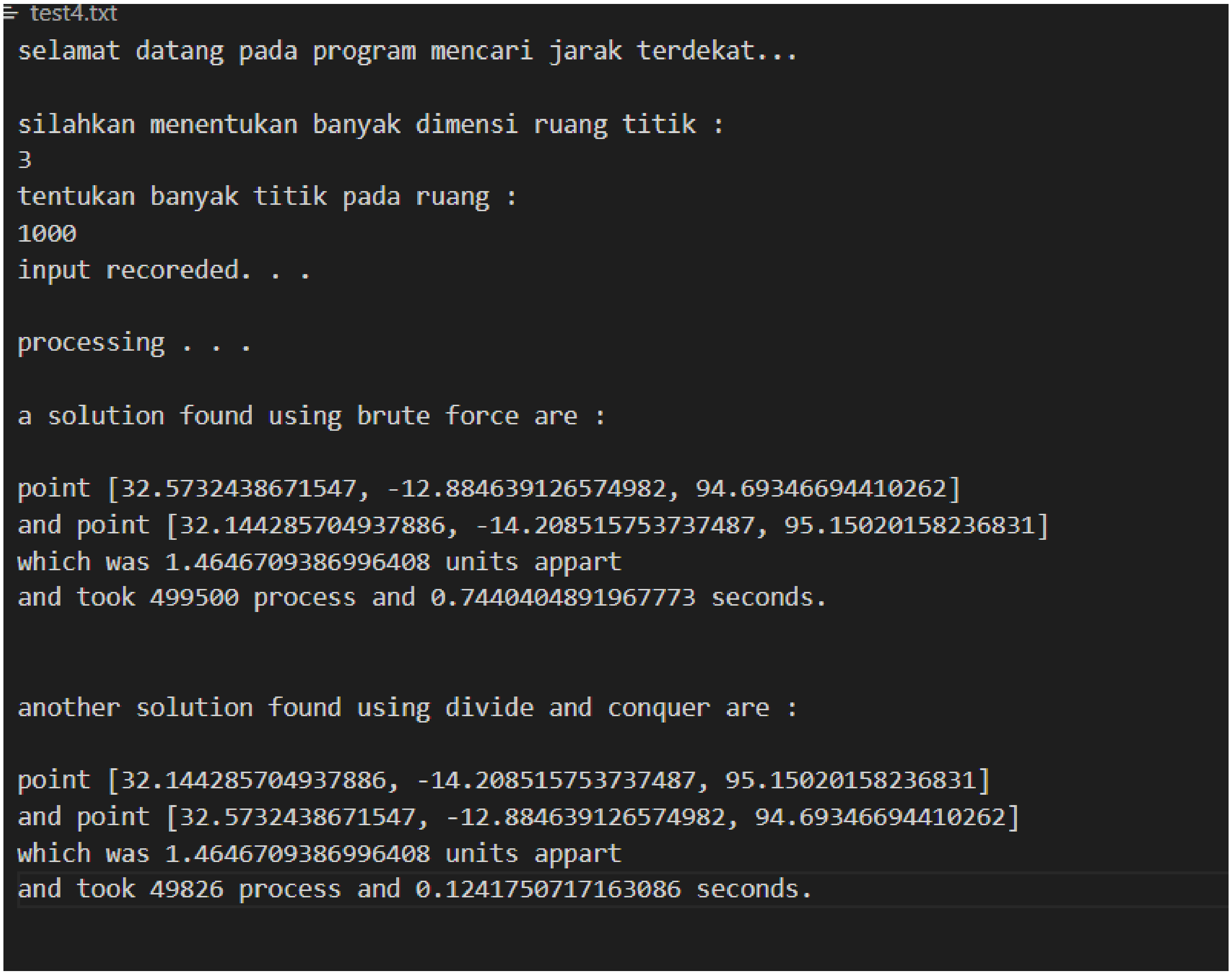
```
selamat datang pada program mencari jarak terdekat...

silahkan menentukan banyak dimensi ruang titik :
3
tentukan banyak titik pada ruang :
64
input recoreded. . .

processing . . .

a solution found using brute force are :

point [-97.90078141844421, -68.32757901511555, -37.28971190795838]
and point [-92.71253318037867, -64.60102489805976, -43.805690758725426]
which was 9.124861957902654 units appart
and took 2016 process and 0.0030562877655029297 seconds.


another solution found using divide and conquer are :

point [-97.90078141844421, -68.32757901511555, -37.28971190795838]
and point [-92.71253318037867, -64.60102489805976, -43.805690758725426]
which was 9.124861957902654 units appart
and took 554 process and 0.0020208358764648438 seconds.
```

3. Testcase 3 (n=128)



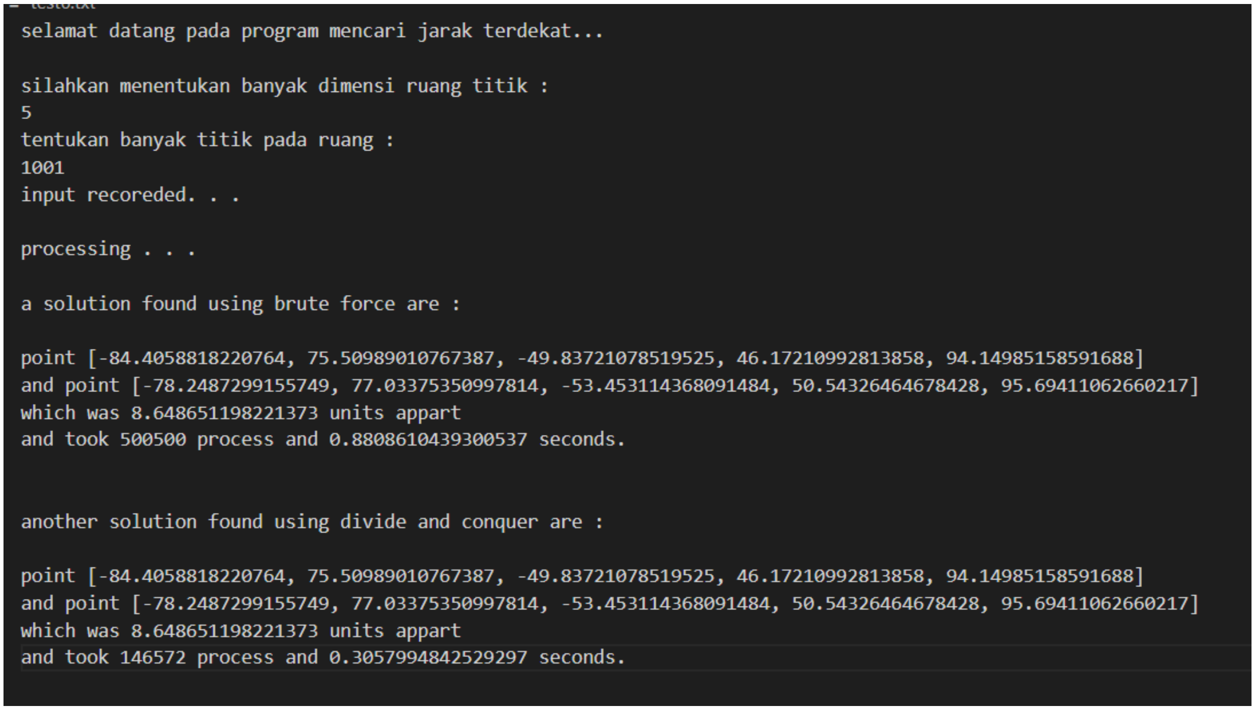
```
selamat datang pada program mencari jarak terdekat...

silahkan menentukan banyak dimensi ruang titik :
3
tentukan banyak titik pada ruang :
128
input recoreded. . .

processing . . .

a solution found using brute force are :

point [-73.22597154106262, 64.21128171552502, 18.344016387697778]
and point [-73.17074295404662, 62.009576037427706, 13.072417992425088]
which was 5.7131696746054486 units appart
and took 8128 process and 0.012088060379002832 seconds.


another solution found using divide and conquer are :

point [-73.22597154106262, 64.21128171552502, 18.344016387697778]
and point [-73.17074295404662, 62.009576037427706, 13.072417992425088]
which was 5.7131696746054486 units appart
and took 1895 process and 0.006018400192260742 seconds.
```

4. Testcase 4 (n=1000)



```
selamat datang pada program mencari jarak terdekat...

silahkan menentukan banyak dimensi ruang titik :
3
tentukan banyak titik pada ruang :
1000
input recoreded. . .

processing . . .

a solution found using brute force are :

point [32.5732438671547, -12.884639126574982, 94.69346694410262]
and point [32.144285704937886, -14.208515753737487, 95.15020158236831]
which was 1.4646709386996408 units appart
and took 499500 process and 0.7440404891967773 seconds.


another solution found using divide and conquer are :

point [32.144285704937886, -14.208515753737487, 95.15020158236831]
and point [32.5732438671547, -12.884639126574982, 94.69346694410262]
which was 1.4646709386996408 units appart
and took 49826 process and 0.1241750717163086 seconds.
```

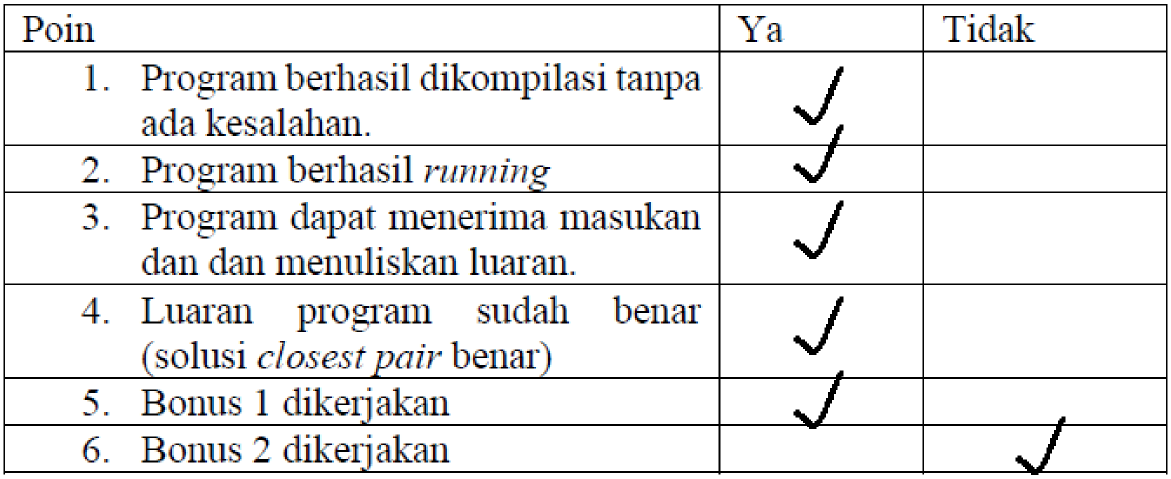
5. Testcase Bonus 1 (dimensi = 5, n=1001)



```
selamat datang pada program mencari jarak terdekat...

silahkan menentukan banyak dimensi ruang titik :
5
tentukan banyak titik pada ruang :
1001
input recoreded. . .

processing . . .

a solution found using brute force are :

point [-84.4058818220764, 75.50989010767387, -49.83721078519525, 46.17210992813858, 94.14985158591688]
and point [-78.2487299155749, 77.03375350997814, -53.453114368091484, 50.54326464678428, 95.69411062660217]
which was 8.648651198221373 units appart
and took 500500 process and 0.8808610439300537 seconds.


another solution found using divide and conquer are :

point [-84.4058818220764, 75.50989010767387, -49.83721078519525, 46.17210992813858, 94.14985158591688]
and point [-78.2487299155749, 77.03375350997814, -53.453114368091484, 50.54326464678428, 95.69411062660217]
which was 8.648651198221373 units appart
and took 146572 process and 0.3057994842529297 seconds.
```

LAMPIRAN

Link Github : https://github.com/tobisns/Tucil2_13521090.git



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		✓