

LAPORAN TUGAS KECIL 2

IF2211 STRATEGI ALGORITMA PENCARIAN TITIK TERDEKAT DENGAN ALGORITMA DIVIDE AND CONQUER

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma
pada Semester 2 (dua) Tahun Akademik 2022/2023.



Oleh:

13521087

Razzan Daksana Yoni

13521095

Muhamad Aji Wibisono

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022**

DAFTAR ISI

DAFTAR ISI	2
BAB I	
PENDAHULUAN	3
1.1. Latar Belakang	3
BAB II	
ALGORITMA	4
2.1. Penjelasan Algoritma Divide and Conquer	4
2.2. Source Program dalam Bahasa Python	6
2.3. Analisis Algoritma	11
BAB III	
EKSPERIMEN	14
BAB IV	
KESIMPULAN	17
LAMPIRAN	18

BAB I

PENDAHULUAN

1.1. Latar Belakang

Algoritma Divide and Conquer adalah algoritma yang menyelesaikan masalah dengan cara membagi permasalahan tersebut menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (*divide*). Setelah dilakukan pembagian, algoritma akan menyelesaikan permasalahan jika sudah berukuran kecil atau kembali membagi secara rekursif jika masih berukuran besar (*conquer*). Di akhir algoritma, solusi masing - masing upa-persoalan akan digabungkan sehingga membentuk solusi permasalahan semula (*combine*).

Algoritma Divide and Conquer dapat menghasilkan solusi dengan kompleksitas yang lebih rendah daripada algoritma brute force dalam beberapa kasus sehingga dapat mempercepat pencarian solusi dalam skala tertentu.

Pada permasalahan pencarian titik terdekat, disajikan sejumlah titik acak dalam suatu ruang berdimensi tertentu. Lalu pada kumpulan titik acak tersebut dilakukan pencarian hingga dihasilkan pasangan titik dengan jarak satu sama lain paling dekat dibandingkan pasangan lainnya.

Pada laporan ini dipaparkan program yang telah dibuat untuk menyelesaikan permasalahan pencarian titik terdekat dengan algoritma Divide and Conquer. Selain itu pada program juga diperlihatkan perbandingan waktu serta kompleksitas penyelesaian masalah yang sama menggunakan algoritma Brute Force sebagai perbandingan. Rincian dari program yang dipaparkan terdapat pada bab - bab selanjutnya.

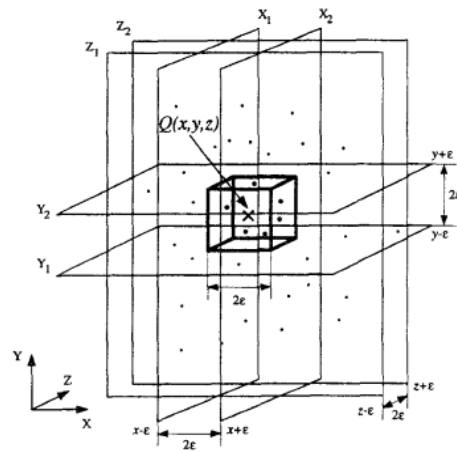
BAB II

ALGORITMA

2.1. Penjelasan Algoritma Divide and Conquer

Pada Algoritma Divide and Conquer yang digunakan ada beberapa langkah yang dilakukan:

1. Kumpulan titik diurutkan berdasarkan salah satu dimensinya.
2. Kumpulan titik akan dibagi menjadi dua ruas sama besar berurut berdasarkan salah satu dimensinya. (*divide*)
3. Jika jumlah titik adalah dua maka didapatkan pasangan titik terdekat, jika jumlah titik adalah tiga maka akan dilakukan pencarian secara Brute Force (*conquer*). Jika lebih dari tiga maka akan dilakukan pembagian lagi sampai didapatkan kumpulan yang berjumlah dua atau tiga.
4. Setelah didapatkan jarak terdekat dari masing - masing ruas akan diperhitungkan jarak antara titik - titik yang berada pada perbatasan ruas satu dan yang lainnya dengan titik yang perbatasan didefinisikan dengan titik yang berjarak kurang dari jarak terdekat yang didapatkan dari kedua belah ruas. (*combine*)
5. Perhitungan pada titik - titik yang berada pada perbatasan ruas satu dan yang lainnya dilakukan dengan melakukan *traversal* terhadap kumpulan lalu membagi titik menjadi dua belah array yang terdapat pada perbatasan untuk yang berada pada satu sisi dan yang lainnya.
6. Setelah terbentuk dua array yang berisikan titik yang berada pada perbatasan pada satu sisi dan yang lainnya, untuk setiap titik pada satu array akan dilakukan pengecekan untuk setiap titik pada array lainnya bahwa titik tersebut memenuhi kriteria pengecekan yaitu jarak pada semua dimensinya kurang dari jarak terdekat yang telah didapatkan sebelumnya sedangkan titik yang tidak memenuhi hanya dilewatkan saja. Ada pula *visualisasi* pada ruang tiga dimensi adalah sebagai berikut:



Sumber: [Closest Point Search in High Dimensions - Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Co \(columbia.edu\)](#)

7. Jika jarak terdekat dari titik - titik perbatasan lebih dekat dari yang didapat sebelumnya, maka jarak terdekat sekarang adalah jarak terdekat dari titik - titik perbatasan.
8. Jarak terdekat akan terus dibandingkan sampai didapat kumpulan titik sejumlah semula. Setelah itu didapatlah jarak terdekat dari sekumpulan titik tersebut.

2.2. Source Program dalam Bahasa Python

Program dibuat dengan antarmuka teks. Selain dapat membuat kumpulan titik secara random, program juga dapat membaca kumpulan titik dari file .txt.

tool.py

```
1 from time import time as currentTime
2 from random import uniform as randomUniform
3
4 def initializeCounter():
5     global euclidN
6     euclidN = 0
7
8 def getDistanceBetweenPoints(point1:list, point2:list, dimension:int = 3) -> float:
9     global euclidN
10    euclidN += 1
11    distance:float = 0
12    for i in range(dimension):
13        distance = distance + ((point1[i] - point2[i]) ** 2)
14    return distance ** 0.5
15
16 def validchecker(point1: list, point2: list, threshold: float, startcheck: int = 0) -> bool:
17     valid = True
18     for i in range(startcheck, len(point1)):
19         if abs(point1[i] - point2[i]) > threshold:
20             valid = False
21             break
22     return valid
23
24 def sortArraym(arrOfPoint:list, key:int, n:int):
25     if(n != 1):
26         arrOfPoint1 = arrOfPoint[:n//2]
27         arrOfPoint2 = arrOfPoint[n//2:]
28
29         len1 = n//2
30         len2 = n - n//2
31
32         sortArraym(arrOfPoint1, key, len1)
33         sortArraym(arrOfPoint2, key, len2)
34
35         idx = 0
36         idx1 = 0
37         idx2 = 0
38
39         while idx1 < len1 and idx2 < len2:
40             if (arrOfPoint1[idx1][key] < arrOfPoint2[idx2][key]):
41                 arrOfPoint[idx] = arrOfPoint1[idx1]
42                 idx1 += 1
43             else:
44                 arrOfPoint[idx] = arrOfPoint2[idx2]
45                 idx2 += 1
46             idx += 1
47         while idx1 < len1:
48             arrOfPoint[idx] = arrOfPoint1[idx1]
49             idx1 += 1
50             idx += 1
51         while idx2 < len2:
52             arrOfPoint[idx] = arrOfPoint2[idx2]
53             idx2 += 1
54             idx += 1
55
56 def sortArrayq(arrOfPoint:list, key:int, i:int, j:int):
57     if (i < j):
58         k = partite2(arrOfPoint, key, i, j)
59         sortArrayq(arrOfPoint, key, i, k-1)
60         sortArrayq(arrOfPoint, key, k+1, j)
```

```

1 #referensi ppt
2 def partite(arrOfPoint:list, key:int, i:int, j:int):
3     pivot = arrOfPoint[i][key]
4
5     #do while?
6     p = i + 1
7     q = j
8
9     while arrOfPoint[p][key] < pivot:
10         p += 1
11         if p > j:
12             p -= 1
13             break
14     while arrOfPoint[q][key] > pivot:
15         q -= 1
16         if q < i:
17             q += 1
18             break
19     arrOfPoint[p], arrOfPoint[q] = arrOfPoint[q], arrOfPoint[p]
20
21     while p < q:
22         while arrOfPoint[p][key] < pivot:
23             p += 1
24             if p > j:
25                 p -= 1
26                 break
27         while arrOfPoint[q][key] > pivot:
28             q -= 1
29             if q < i:
30                 q += 1
31                 break
32         arrOfPoint[p], arrOfPoint[q] = arrOfPoint[q], arrOfPoint[p]
33
34     arrOfPoint[p], arrOfPoint[q] = arrOfPoint[q], arrOfPoint[p]
35     arrOfPoint[i], arrOfPoint[q] = arrOfPoint[q], arrOfPoint[i]
36
37     return q
38
39 #referensi internet
40 def partite2(arrOfPoint:list, key:int, i:int, j:int):
41     pivot = arrOfPoint[j][key]
42     p = i-1
43
44     for l in range(i, j):
45         if arrOfPoint[l][key] <= pivot:
46             p += 1
47             arrOfPoint[p], arrOfPoint[l] = arrOfPoint[l], arrOfPoint[p]
48
49     p += 1
50     arrOfPoint[p], arrOfPoint[j] = arrOfPoint[j], arrOfPoint[p]
51     return p
52
53 def readFile(fileName:str) -> tuple :
54     fileName = '../test/' + fileName
55
56     arrOfPoints = []
57     dimension:int
58     n:int
59
60     with open(fileName, 'r') as f:
61         for line in f:
62             temp = line.split(' ')
63             point = []
64             dimension = len(temp)
65
66             for num in temp:
67                 if num != '\n':
68                     point = point + [float(num)]
69             arrOfPoints = arrOfPoints + [point]
70
71     f.close()
72
73     n = len(arrOfPoints)
74     return (arrOfPoints, n, dimension)

```

visualization.py

```
1 from matplotlib.pyplot import figure, show
2
3 def three_dimensional_plotting(arrOfPoint:list, pointA:list, pointB:list, name:str = "") :
4     fig = figure(num=name)
5
6     # 3D plot from array of points
7     ax = fig.add_subplot(111, projection='3d')
8     for point in arrOfPoint:
9         ax.scatter(point[0], point[1], point[2], color='blue')
10
11     ax.scatter(pointA[0], pointA[1], pointA[2], color='red')
12     ax.scatter(pointB[0], pointB[1], pointB[2], color='red')
13     show()
14
```

bruteForce.py

[illegible]

divideNConquer.py

```
1 from bruteForce import getClosestPairByBruteForce
2 from tools import getDistanceBetweenPoints, validchecker
3
4 def getClosestPairByDivideNConquer(arrOfPoint:list, n:int, dimension:int = 3) -> tuple :
5     # I.S. arrOfPoint is presorted to x(dimension 0)
6     if n <= 3: # if there are 3 or less points, use brute force
7         return getClosestPairByBruteForce(arrOfPoint, n, dimension)
8     else :
9         # divide array of points into 2 sub arrays
10        arrOfPoint1 = arrOfPoint[:n // 2]
11        arrOfPoint2 = arrOfPoint[n // 2:]
12
13        # find closest pair in each sub array
14        pointA1, pointB1, minDistance1 = getClosestPairByDivideNConquer(arrOfPoint1, n // 2, dimension)
15        pointA2, pointB2, minDistance2 = getClosestPairByDivideNConquer(arrOfPoint2, n - (n // 2),
16        dimension)
17
18        # find min distance
19        pointA, pointB, minDistance = (pointA1, pointB1, minDistance1) if (minDistance1 < minDistance2)
20        else (pointA2, pointB2, minDistance2)
21
22        # find closest pair that cross the middle line
23        # find middle line
24        middleLine = (arrOfPoint[n // 2 - 1][0] + arrOfPoint[n // 2][0]) / 2
25
26        nMiddleLine1 = 0
27        nMiddleLine2 = 0
28        # find points that are in the middle line respect to x
29        arrOfPointInMiddleLine1 = []
30        arrOfPointInMiddleLine2 = []
31
32        for i in range(n):
33            val = arrOfPoint[i][0] - middleLine
34            if val < minDistance and val >= 0:
35                arrOfPointInMiddleLine1 = arrOfPointInMiddleLine1 + [arrOfPoint[i]]
36                nMiddleLine1 += 1
37            elif val > (-1)*minDistance and val <= 0:
38                arrOfPointInMiddleLine2 = arrOfPointInMiddleLine2 + [arrOfPoint[i]]
39                nMiddleLine2 += 1
40
41        ''' Alternative
42        # sort array of points by y coordinate
43        #sortArrayq(arrOfPointInMiddleLine1, 1, 0, nMiddleLine1-1)
44        #sortArrayq(arrOfPointInMiddleLine2, 1, 0, nMiddleLine2-1)
45
46        # find closest pair that cross the middle line
47        #k = 0
48        for i in range(nMiddleLine1):
49            j = 0 #j = k
50            while j < nMiddleLine2:
51                #if arrOfPointInMiddleLine2[j][1] - arrOfPointInMiddleLine1[i][1] > minDistance:
52                #    break
53                #if arrOfPointInMiddleLine2[j][1] - arrOfPointInMiddleLine1[i][1] < (-1)*minDistance:
54                #    k = j + 1
55                if validchecker(arrOfPointInMiddleLine1[i], arrOfPointInMiddleLine2[j], minDistance, 1):
56                    distance = getDistanceBetweenPoints(arrOfPointInMiddleLine1[i],
57                    arrOfPointInMiddleLine2[j], dimension)
58                    if distance < minDistance:
59                        minDistance = distance
60                        pointA = arrOfPointInMiddleLine1[i]
61                        pointB = arrOfPointInMiddleLine2[j]
62                    j += 1
63
64        '''nPointToCheck = (6) if (nMiddleLine - i - 1 > (6)) else nMiddleLine - i - 1
65        # nPointToCheck = nMiddleLine
66
67        for j in range(1, nPointToCheck+1):
68            if abs(arrOfPointInMiddleLine[i][1] - arrOfPointInMiddleLine[i+j][1]) < minDistance :
69                distance = getDistanceBetweenPoints(arrOfPointInMiddleLine[i], arrOfPointInMiddleLine
70                [i+j], dimension)
71                if distance < minDistance:
72                    minDistance = distance
73                    pointA = arrOfPointInMiddleLine[i]
74                    pointB = arrOfPointInMiddleLine[i+j]'''
75
76        return (pointA, pointB, minDistance)
```

main.py

```
1 from bruteForce import getClosestPairByBruteForce
2 from divideNConquer import getClosestPairByDivideNConquer
3 from visualization import three_dimensional_plotting
4 import tools
5 from sys import setrecursionlimit
6
7 def main():
8     setrecursionlimit(100000)
9     exit = False
10    while not(exit) :
11        inputUser = -1
12        print("""
13 Input action:
14 1. Random
15 2. Read File
16 3. exit""")
17        try:
18            inputUser = int(input("-> "))
19            if inputUser == 1 :
20                n = int(input("Enter number of points: "))
21                if n < 2:
22                    raise Exception("Number of points must be larger than 1")
23                dimension = int(input("Enter dimension: "))
24                if dimension < 1:
25                    raise Exception("Dimension must be larger than 0")
26                arrOfPoint = []
27                for i in range(n):
28                    temp = [tools.randomUniform(-100, 100) for j in range(dimension)]
29                    arrOfPoint = arrOfPoint + [temp]
30            elif inputUser == 2 :
31                fileName = input("Input File Name: ")
32                arrOfPoint, n, dimension = tools.readFile(fileName)
33            elif inputUser == 3 :
34                exit = True
35            else :
36                pass
37        except Exception as exception:
38            print(exception)
39            inputUser = -1
40        if ((inputUser == 1) or (inputUser == 2)) :
41            tools.initializeCounter()
42            timeBruteForce = tools.currentTime()
43            point1BF, point2BF, minDistanceBF = getClosestPairByBruteForce(arrOfPoint, n,
44 dimension)
45            timeBruteForce = tools.currentTime() - timeBruteForce
46            print("Closest pair is Using Brute Force: ")
47            print("Point A: ", point1BF)
48            print("Point B: ", point2BF)
49            print("Distance: ", minDistanceBF)
50            print("Time: ", timeBruteForce)
51            print("Times euclidean distance called: ", tools.euclidN)
52            tools.initializeCounter()
53            timeDivideNConquer = tools.currentTime()
54            point1DC, point2DC, minDistanceDC= getClosestPairByDivideNConquer(arrOfPoint, n,
55 dimension)
56            timeDivideNConquer = tools.currentTime() - timeDivideNConquer
57            print("Closest pair is Using Divide and Conquer: ")
58            print("Point A: ", point1DC)
59            print("Point B: ", point2DC)
60            print("Distance: ", minDistanceDC)
61            print("Time: ", timeDivideNConquer)
62            print("Times euclidean distance called: ", tools.euclidN)
63            print()
64            if ((1 < n <= 1000) and (dimension == 3)):
65                three_dimensional_plotting(arrOfPoint, point1BF, point2BF, "Brute Force")
66                three_dimensional_plotting(arrOfPoint, point1DC, point2DC, "Divide and Conquer")
67
68 if __name__ == '__main__':
69     main()
```

2.3. Analisis Algoritma

Algoritma Brute Force memiliki kompleksitas diperhitungkan secara sederhana dengan jumlah *loop* yang terdapat yaitu $O(n^2)$ untuk semua kasus yaitu dengan cara memilih semua pasangan titik lalu memanggil perhitungan *euclidean distance* dalam tiap *loop*.

Di sisi lain, Algoritma Divide and Conquer memiliki kompleksitas yang tidak sesederhana algoritma Brute Force. Merujuk kepada *source code* yang tertera pada bagian 2.2, algoritma Divide and Conquer memerlukan kumpulan titik untuk diurutkan terlebih dahulu sebelum dilakukan Divide and Conquer. Pengurutan yang diimplementasikan, Quick Sort, memiliki kompleksitas $O(n \log n)$. Setelah itu, algoritma Divide and Conquer memiliki rekursif yang memanggil dirinya sendiri dengan jumlah n dibagi kurang lebih sama rata menjadi dua bagian sehingga menghasilkan $T(\frac{n}{2})$. Untuk n yang bernilai 2, maka kompleksitas adalah 1 dan untuk n yang bernilai 3 kompleksitas adalah 3, ini menghasilkan notasi $O(1)$. Saat penggabungan dilakukan traversal kembali untuk kedua kumpulan titik yang berada pada perbatasan. Pada kemungkinan terburuk, semua titik berada pada perbatasan sehingga pemilihan yang dilakukan adalah $(\frac{n}{2})^2$ didapatkan notasi yaitu $O(n^2)$. Secara kasus terburuk, algoritma Divide and Conquer memiliki kompleksitas:

$$T(n) = 1, n = 2$$

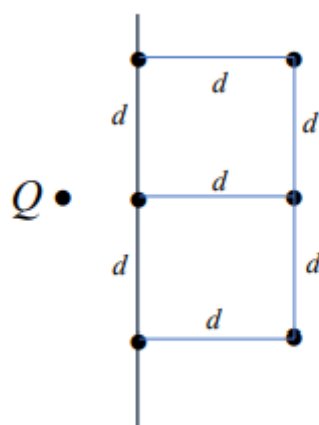
$$T(n) = 3, n = 3$$

$$T(n) = 2(T(n/2)) + O(n^2), n > 3$$

Menggunakan teorema master dengan $a = 2$, $b = 2$, dan $d = 2$, didapat kompleksitas pada algoritma Divide and Conquer setelah ditambahkan kompleksitas pengurutan adalah $O(n^2 + n \log n) = O(n^2)$.

Akan tetapi, perhitungan kompleksitas yang dilakukan pada algoritma Divide and Conquer sebelumnya berbeda dengan perhitungan kompleksitas yang dilakukan pada algoritma Brute Force. Perbedaan terbesar adalah bahwa pada algoritma Brute Force, kompleksitas algoritma berlaku untuk perhitungan *euclidean distance*, sedangkan pada algoritma Divide and Conquer kompleksitas hanya berlaku untuk pemilihan titik. Perlu diketahui bahwa perhitungan *euclidean distance* memakai *resource* yang jauh lebih banyak daripada pemilihan titik karena perhitungan *euclidean distance* dilakukan dengan perkalian untuk setiap dimensi diikuti dengan pertambahan dan pengakaran sedangkan pemilihan titik hanya menggunakan perbandingan untuk setiap dimensi.

Jika dilakukan perhitungan ulang kompleksitas menggunakan jumlah pemanggilan *euclidean distance* saja, algoritma Divide and Conquer akan memiliki kompleksitas yang berbeda. Sorting menjadi tidak diperhitungkan karena tidak melakukan perhitungan *euclidean distance*. Perhitungan rekursif yang dilakukan selanjutnya memiliki kompleksitas yang sama seperti sebelumnya. Perbedaan besar muncul ketika dilakukan penggabungan antar ruas. Sebab jarak minimum antara dua titik pada kedua belah sisi sudah diketahui sebelumnya, dapat ditentukan jumlah maksimum titik yang berada pada cakupan pengecekan. Ada pula visualisasi pada dimensi ruang dua dimensi adalah sebagai berikut:



Sumber: [Algoritma Divide and Conquer \(itb.ac.id\)](http://itb.ac.id)

Dengan melakukan *scaling-up* pada dimensi yang lebih tinggi dan induksi secara matematis dapat ditentukan bahwa jumlah titik maksimum yang mungkin memenuhi kriteria pengecekan mengikuti persamaan $2 \times 3^{d-1}$ dengan d adalah jumlah dimensi ruang titik - titik tersebut berada. Dari perhitungan tersebut dapat terlihat bahwa perhitungan *euclidean distance* pada algoritma Divide and Conquer bukan dipengaruhi oleh n atau jumlah titik, melainkan jumlah dimensi ruang titik - titik tersebut berada. Dengan demikian, kompleksitas dari perhitungan *euclidean distance* titik - titik yang berada pada perbatasan pembagian kedua ruas untuk kemungkinan terburuk adalah $(\frac{n}{2}) \times (2 \times 3^{d-1})$, sehingga didapat notasi Big O adalah $O(n)$.

Secara kasus terburuk, algoritma Divide and Conquer memiliki kompleksitas untuk perhitungan *euclidean distance*:

$$T(n) = 1, n = 2$$

$$T(n) = 3, n = 3$$

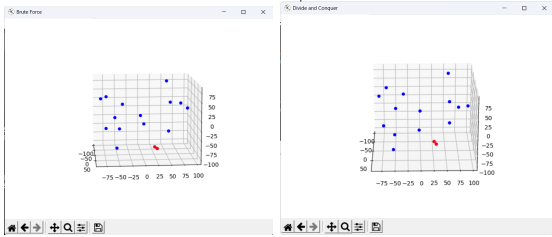
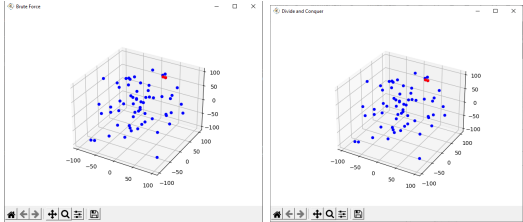
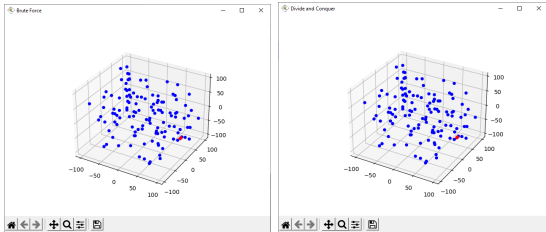
$$T(n) = 2(T(n/2)) + O(n), n > 3$$

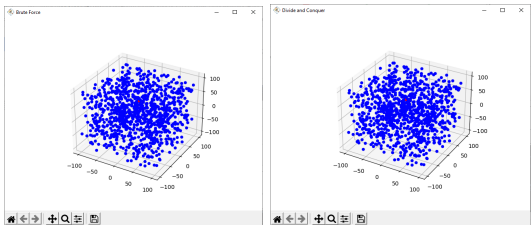
Menggunakan teorema master dengan $a = 2$, $b = 2$, dan $d = 1$, didapat kompleksitas untuk kemungkinan terburuk perhitungan *euclidean distance* pada algoritma Divide and Conquer adalah $O(n \log n)$. Walaupun begitu jika perhitungan *euclidean distance* antara kedua ruas tidak akan melebihi jumlah titik secara total dan tidak akan perhitungan tidak akan dilakukan melebihi perhitungan yang dilakukan oleh Brute Force.

Dari analisis yang telah dilakukan, dapat disimpulkan bahwa algoritma Divide and Conquer relatif lebih sangkil dibandingkan algoritma Brute Force untuk jumlah titik yang tinggi dengan kompleksitas algoritma Divide and Conquer adalah $O(n \log n)$ dan kompleksitas algoritma Brute Force adalah $O(n^2)$. Hal ini dibuktikan dengan eksperimen yang telah dilakukan pada bab selanjutnya.

BAB III

EKSPERIMEN

Test Case	Gambar	Keterangan
n = 16	<pre> Enter number of points: 16 Enter dimension: 3 Closest pair is Using Brute Force: Point A: [-12.73394377291855, 32.44806411571915, -98.55656510964073] Point B: [-24.99325172270295, 28.078090139415792, -97.30978303442244] Distance: 13.074470142399552 Time: 0.0 Times euclidean distance called: 120 Closest pair is Using Divide and Conquer: Point A: [-24.99325172270295, 28.078090139415792, -97.30978303442244] Point B: [-12.73394377291855, 32.44806411571915, -98.55656510964073] Distance: 13.074470142399552 Time: 0.0 Times euclidean distance called: 14 </pre> 	<p>Untuk n = 16 algoritma Divide and Conquer lebih cepat pada kasus ini banyak perhitungan <i>euclidean distance</i> mendekati $O(n)$.</p>
n = 64	<pre> Enter number of points: 64 Enter dimension: 3 Closest pair is Using Brute Force: Point A: [35.90967874293787, 53.54417693019164, 91.62801341022376] Point B: [41.03182081261835, 56.88348457361835, 88.04738620613264] Distance: 7.085774911335809 Time: 0.00400090217590332 Times euclidean distance called: 2016 Closest pair is Using Divide and Conquer: Point A: [41.03182081261835, 56.88348457361835, 88.04738620613264] Point B: [35.90967874293787, 53.54417693019164, 91.62801341022376] Distance: 7.085774911335809 Time: 0.001001596450805664 Times euclidean distance called: 67 </pre> 	<p>Untuk n = 64 algoritma Divide and Conquer lebih cepat pada kasus ini banyak perhitungan <i>euclidean distance</i> mendekati $O(n)$.</p>
n = 128	<pre> Enter number of points: 128 Enter dimension: 3 Closest pair is Using Brute Force: Point A: [84.3078161068334, 28.365499291675974, -64.83391304625317] Point B: [83.67399344548647, 24.603186531327808, -64.82322039137594] Distance: 3.815343078359253 Time: 0.013014078140258789 Times euclidean distance called: 8128 Closest pair is Using Divide and Conquer: Point A: [83.67399344548647, 24.603186531327808, -64.82322039137594] Point B: [84.3078161068334, 28.365499291675974, -64.83391304625317] Distance: 3.815343078359253 Time: 0.003019571304321289 Times euclidean distance called: 123 </pre> 	<p>Untuk n = 128 algoritma Divide and Conquer lebih cepat pada kasus ini banyak perhitungan <i>euclidean distance</i> mendekati $O(n)$.</p>

<p>$n = 1000$</p>	<pre> Enter number of points: 1000 Enter dimension: 3 Closest pair is Using Brute Force: Point A: [-13.186247673668959, 22.191941771696747, 29.046558424409056] Point B: [-13.15761928228332, 22.202211619368057, 28.341580964323896] Distance: 0.7056332431175882 Time: 0.8585119247436523 Times euclidean distance called: 499500 Closest pair is Using Divide and Conquer: Point A: [-13.15761928228332, 22.202211619368057, 28.341580964323896] Point B: [-13.186247673668959, 22.191941771696747, 29.046558424409056] Distance: 0.7056332431175882 Time: 0.034818754959106445 Times euclidean distance called: 1016 </pre> 	<p>Untuk $n = 1000$ algoritma Divide and Conquer lebih cepat pada kasus ini banyak perhitungan <i>euclidean distance</i> mendekati $O(n)$.</p>
<p>$n = 100000$</p>	<pre> Enter number of points: 100000 Enter dimension: 2 Closest pair is Using Brute Force: Point A: [7.278110760959322, 2.8372973526930707] Point B: [7.278751969374824, 2.837525401458379] Distance: 0.000680554532325429 Time: 2992.341346025467 Times euclidean distance called: 4999950000 Closest pair is Using Divide and Conquer: Point A: [7.278110760959322, 2.8372973526930707] Point B: [7.278751969374824, 2.837525401458379] Distance: 0.000680554532325429 Time: 1.4180035591125488 Times euclidean distance called: 123762 </pre>	<p>Untuk $n = 100000$ algoritma Divide and Conquer lebih cepat pada kasus ini banyak perhitungan <i>euclidean distance</i> mendekati $O(n)$.</p> <p>Visualisasi tidak dapat dihasilkan karena jumlah titik yang terlalu banyak.</p>
<p>$n = 128$ dimensi = 20</p>	<pre> Enter number of points: 128 Enter dimension: 20 Closest pair is Using Brute Force: Point A: [53.1563956999985, 7.096347495876245, 23.413462008764313, 27.377628262910775, 48.13188310167044, -30.5604769899481, 28.281656476846223, 86.61751726000978, 15.409080988292814, 71.6501055926619, 13.636884130519377, -96.8924967734608, -91.03032210032968, -28.376391462364836, 32.30330838882995, -38.66670372747401, -93.33282577906061, -23.938055537182663, -84.82845602856875, 3.3985871871704063] Point B: [67.1170478464397, 30.830953443571502, 33.64178444892306, 90.64453277541657, -15.189342762438201, -33.29074435994022, 26.496296082711197, 46.32092304517809, -21.266393582735915, 51.22587849014758, 33.466143794975466, 12.917827184058225, -55.850788788309515, -12.98431697668974, 4.074889830302581, -2.765349603702333, -45.09087630045445, -37.17850885245777, -98.08156410258641, 60.62808142862389] Distance: 185.0537103352395 Time: 0.05797839164733887 Times euclidean distance called: 8128 Closest pair is Using Divide and Conquer: Point A: [67.1170478464397, 30.830953443571502, 33.64178444892306, 90.64453277541657, -15.189342762438201, -33.29074435994022, 26.496296082711197, 46.32092304517809, -21.266393582735915, 51.22587849014758, 33.466143794975466, 12.917827184058225, -55.850788788309515, -12.98431697668974, 4.074889830302581, -2.765349603702333, -45.09087630045445, -37.17850885245777, -98.08156410258641, 60.62808142862389] Point B: [53.1563956999985, 7.096347495876245, 23.413462008764313, 27.377628262910775, 48.13188310167044, -30.5604769899481, 28.281656476846223, 86.61751726000978, 15.409080988292814, 71.6501055926619, 13.636884130519377, -96.8924967734608, -91.03032210032968, -28.376391462364836, 32.30330838882995, -38.66670372747401, -93.33282577906061, -23.938055537182663, -84.82845602856875, 3.3985871871704063] Distance: 185.0537103352395 Time: 0.0799229431152344 Times euclidean distance called: 7548 </pre>	<p>Banyaknya jumlah titik juga relatif terhadap jumlah dimensi, untuk jumlah titik yang kecil dengan jumlah dimensi yang tinggi, jumlah perhitungan <i>euclidean distance</i> yang dilakukan algoritma Divide and Conquer juga menjadi besar.</p> <p>Visualisasi tidak dapat dihasilkan karena dimensi terlalu tinggi.</p>

<p>n = 1000</p> <p>dimensi = 20</p>	<pre> Enter number of points: 1000 Enter dimension: 20 Closest pair is Using Brute Force: Point A: [-38.82876669020791, -58.12938036632218, -36.53854281486877, 26.588315589 715165, -88.12149420914221, -11.038011792655539, -13.613759524638809, 18.3464165866 09024, 9.199710269817189, 48.0204432892248, 65.11228451173454, 57.12505762082901, - 62.22496864573228, -81.62311968506519, -78.07004798909507, 24.317436809819327, 0.19 810848864267427, 18.547930608020806, -7.672841162405518, -1.8432518388085413] Point B: [-47.4057776019041, 31.12732408601815, -94.00366300347824, 18.76423333374 4613, -87.14536070106058, -19.106293785108335, -11.092803237203404, -28.64690982398 143, -18.169944089863677, 62.1664854860214, 68.00465836749457, 66.68107382185494, - 65.39300502851279, -96.59001706902528, -91.59739431421899, 10.829316079835166, 10.4 2954954716653, 42.95434388129374, -5.829921134677392, -68.28508702324206] Distance: 142.99491978564333 Time: 3.6621642112731934 Times euclidean distance called: 499500 Closest pair is Using Divide and Conquer: Point A: [-38.82876669020791, -58.12938036632218, -36.53854281486877, 26.588315589 715165, -88.12149420914221, -11.038011792655539, -13.613759524638809, 18.3464165866 09024, 9.199710269817189, 48.0204432892248, 65.11228451173454, 57.12505762082901, - 62.22496864573228, -81.62311968506519, -78.07004798909507, 24.317436809819327, 0.19 810848864267427, 18.547930608020806, -7.672841162405518, -1.8432518388085413] Point B: [-47.4057776019041, 31.12732408601815, -94.00366300347824, 18.76423333374 4613, -87.14536070106058, -19.106293785108335, -11.092803237203404, -28.64690982398 143, -18.169944089863677, 62.1664854860214, 68.00465836749457, 66.68107382185494, - 65.39300502851279, -96.59001706902528, -91.59739431421899, 10.829316079835166, 10.4 2954954716653, 42.95434388129374, -5.829921134677392, -68.28508702324206] Distance: 142.99491978564333 Time: 2.2693347930908203 Times euclidean distance called: 157431 </pre>	<p>Jumlah dimensi juga mempengaruhi perhitungan karena relatif jumlah terhadap dimensi seperti telah disebutkan dalam sub bab sebelumnya.</p> <p>Visualisasi tidak dapat dihasilkan karena dimensi terlalu tinggi.</p>
-------------------------------------	--	--

BAB IV

KESIMPULAN

Pencarian dua titik terdekat dalam R^n dapat diselesaikan dengan algoritma Brute Force dan algoritma Divide and Conquer. Dengan mengimplementasikan algoritma Divide and Conquer yang penulis tawarkan didapatkan jumlah perhitungan *euclidean distance* lebih sedikit untuk dimensi rendah dan jumlah titik banyak dibandingkan algoritma Brute Force. Dan didapatkan juga jumlah titik relatif terhadap jumlah dimensi yaitu jumlah titik yang kecil dengan jumlah dimensi yang tinggi membuat jumlah perhitungan *euclidean distance* pada algoritma Divide and Conquer juga menjadi besar.

LAMPIRAN

Link Spesifikasi Tugas

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tucil2-Stima-2023.pdf>

Cek List Spesifikasi

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

Repository Github

Berikut adalah link repository GitHub untuk program penulis.

https://github.com/razzanYoni/Tucil2_13521087_13521095