IF2211
Strategi Algoritma

LAPORAN TUGAS KECIL STRATEGI ALGORITMA

PEMANFAATAN ALGORITMA DIVIDE AND CONQUER DALAM PENCARIAN PASANGAN TITIK TERDEKAT

Kelas Mahasiswa K03

Dosen: Ir. Rila Mandala, M.Eng., Ph.D.

Diajukan sebagai tugas kecil Mata Kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2022/2023

Disusun Oleh:

Muhamad Salman Hakim Alfarisi	**13521010**
Muhhamad Syauqi Jannatan	**13521014**

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

IF2211
Strategi Algoritma

DAFTAR ISI

BAB I

DESKRIPSI MASALAH

Mencari pasangan titik terdekat dengan Algoritma Divide and Conquer pada bidang 3D. Misalkan terdapat n buah titik pada ruang 3D. Setiap titik P di dalam ruang dinyatakan dengan koordinat P = (x, y, z). Tujuannya adalah mencari sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titik P1 = (x1, y1, z1) dan P2 = (x2, y2, z2) dihitung menggunakan rumus Euclidean dan dinyatakan dalam bentuk jarak Euclidean.

BAB II

DASAR TEORI

Algoritma divide and conquer adalah metode yang digunakan untuk memecahkan masalah yang kompleks dengan membaginya menjadi sub-masalah yang lebih kecil, kemudian menyelesaikan sub-masalah tersebut secara rekursif, dan menggabungkan solusi sub-masalah tersebut.

Terdapat tiga langkah utama dalam algoritma divide and conquer:

1. Divide (Pembagian): Langkah pertama dalam algoritma divide and conquer adalah membagi masalah asli menjadi sub-masalah yang lebih kecil dan terpisah. Proses pembagian ini biasanya dilakukan secara rekursif hingga sub-masalah tersebut menjadi cukup kecil untuk dapat diselesaikan dengan mudah.

2. Conquer (Penaklukan): Langkah kedua dalam algoritma divide and conquer adalah menyelesaikan setiap sub-masalah secara terpisah. Setiap sub-masalah ini biasanya diselesaikan dengan menggunakan algoritma yang sama seperti algoritma asli, atau menggunakan teknik lain yang sesuai untuk menyelesaikan sub-masalah tersebut.

3. Combine (Penggabungan): Langkah terakhir dalam algoritma divide and conquer adalah menggabungkan solusi dari setiap sub-masalah untuk membentuk solusi akhir dari masalah asli. Proses penggabungan ini biasanya melibatkan penggabungan solusi dari setiap sub-masalah secara teratur hingga terbentuk solusi akhir yang lengkap.untuk membentuk solusi akhir dari masalah asli.

BAB III

IMPLEMENTASI PROGRAM

3.1 Algoritma

Program yang kami buat memiliki langkah-langkah sebagai berikut:

1. Program meminta input dari user berupa jumlah titik yang ingin dihasilkan dan jumlah dimensi dari titik tersebut.
2. Program akan menghasilkan titik-titik yang telah di acak. Langkah ini diimplementasikan pada fungsi ***generate_points(n, m)***.
3. Apabila jumlah titik(n) adalah dua, program akan mengembalikan nilai jarak antara kedua titik tersebut. Apabila jumlah titik(n) adalah tiga, program akan menghitung jarak setiap kombinasi titik dan mengembalikan jarak pasangan titik terdekat. Langkah ini diimplementasikan pada fungsi ***closest_pair(points)***.
4. Apabila jumlah titik(n) adalah lebih dari tiga, program akan mengurutkan titik-titik tersebut berdasarkan absis, kemudian akan memecah menjadi dua bagian, yaitu *left* dan *right*. Setiap bagian akan dipecah lagi hingga tersisa dua atau tiga titik, lalu dihitung menggunakan fungsi rekursif ***closest_pair(points)*** untuk mencari pasangan titik dengan jarak terdekat.
5. Hasil dari dua bagian akan dibandingkan untuk mencari jarak yang terkecil lalu disimpan dalam variabel *d*.
6. Langkah terakhir adalah memeriksa keberadaan titik yang lebih dekat yang terletak di setiap sisi titik tengah. Untuk ini, algoritma mengiterasi semua titik pada himpunan asli dan memeriksa apakah ada titik dalam jarak *d* (jarak dari pasangan terdekat yang ditemukan sejauh ini) dari titik tengah. Jika ada titik seperti itu, maka algoritma membandingkan jarak antara titik tersebut dan semua titik lain yang terletak dalam jarak *d* dari titik tengah. Jika ditemukan pasangan yang lebih dekat, maka pasangan terdekat dan jaraknya diperbarui. Langkah ini diimplementasikan menggunakan fungsi ***s_strip(points)***.

7. Program mengembalikan pasangan titik terdekat dan jaraknya.

3.2 Source Code Program

divideConquere.py

```
import matplotlib.pyplot as plt
import random
import math
counterEuclidean = 0

def generate_points(n, m):
  #complexity O(n)
  points = []
  for i in range(n):
      temp = []
      for j in range(m):
          temp.append(random.randint(0,1000))
      points.append(temp)
  return points

def distance(p1,p2):
  #complexity O(1)
  global counterEuclidean
  counterEuclidean+=1
  result = float(0)
  for i in range(len(p1)):
      result += (p1[i]-p2[i])**2
  return math.sqrt(result)

def closest_pair(points):
  #complexity O(nlogn)
  if(len(points) == 2):
    return points[0], points[1], distance(points[0], points[1])
  elif(len(points) == 3):
    d1 = distance(points[0], points[1])
    d2 = distance(points[0], points[2])
    d3 = distance(points[1], points[2])
    if(d1 < d2 and d1 < d3):
      return points[0], points[1], d1
    elif(d2 < d1 and d2 < d3):
```

```
      return points[0], points[2], d2
    else:
      return points[1], points[2], d3
  else:
    return s_strip(points)

def s_strip(points):
  #complexity O(nlogn)
  points.sort(key = lambda x: x[0])
  if(len(points) % 2 == 0):
    mid = int(len(points)/2)
  else:
    mid = int(len(points)//2) + 1
  left = points[:mid]
  right = points[mid:]
  p1, q1, d1 = closest_pair(left)
  p2, q2, d2 = closest_pair(right)
  if(d1 < d2):
    d = d1
    p = p1
    q = q1
  else:
    d = d2
    p = p2
    q = q2
  for i in range(len(points)):
    if(abs(points[i][0] - points[mid][0]) < d):
      for j in range(i+1, len(points)):
        if(abs(points[j][0] - points[mid][0]) < d):
          if(distance(points[i], points[j]) < d):
            d = distance(points[i], points[j])
            p = points[i]
            q = points[j]
  return p, q, d
```

IF2211
Strategi Algoritma

bruteForce.py

```
import divideConquer as dc

def closest_pair_bf(points):
  #complexity O(n^2)
  # brute force
  d = dc.distance(points[0], points[1])
  p = points[0]
  q = points[1]
  for i in range(len(points)):
    for j in range(i+1, len(points)):
      if(dc.distance(points[i], points[j]) < d):
        d = dc.distance(points[i], points[j])
        p = points[i]
        q = points[j]
  return p, q, d
```

main.py

```
import divideConquer as dc
import bruteForce as bf
import visualize as vs
import platform
import time
import os
def splashScreen():
    print (" _______     __          _______     _______     _______     _______
_____      _______   ______      __     ______     ")
    print ("/\   ___\   /\ \        /\  __ \   /\  ___\   /\  ___\   /\
___\   /\__  _\    /\  == \ /\  __ \   /\ \   /\  == \   ")
    print ("\ \ \____  \ \ \____  \ \ \/\ \  \ \___  \  \ \ \  __\   \ \___
\  \/_/\ \/    \ \  _-/ \ \  __ \  \ \ \  \ \  __<   ")
    print (" \ \_____\  \ \_____\  \ \_____\  \/\_____\  \ \_____\
\/\_____\    \ \_\     \ \_\    \ \_\ \_\  \ \_\  \ \_\ \_\ ")
    print ("  \/_____/   \/_____/   \/_____/   \/_____/   \/_____/
\/_____/     \/_/      \/_/     \/_/\/_/   \/_/   \/_/ /_/ ")
```

```
    print ("
")

def main():
    splashScreen()
    n = int(input("Masukkan jumlah titik: "))
    m = int(input("Masukkan jumlah dimensi: "))
    while(n < 2):
        n = int(input("Masukkan jumlah titik: "))
        m = int(input("Masukkan jumlah dimensi: "))

    points = dc.generate_points(n, m)
    os.system('cls' if os.name == 'nt' else 'clear')
    print("============================================================")
    print("                 Divide & Conquer Algorithm                 ")
    print("============================================================")
    startTime1 = time.time()
    p, q, d = dc.closest_pair(points)
    endTime1 = time.time()
    totalTime1 = (endTime1 - startTime1)*1000
    print("Waktu eksekusi divide and conquer: {:.3f}".format(totalTime1),
"ms")
    print("Jumlah operasi euclidean: ", dc.counterEuclidean)
    print("Jarak terdekat: {:.3f}".format(d))
    print("Platform: ", platform.processor())
    print("============================================================")
    print("                   Brute Force Algorithm                    ")
    print("============================================================")

    startTime2 = time.time()
    p1, q1, d1 = bf.closest_pair_bf(points)
    endTime2 = time.time()
    totalTime2 = (endTime2 - startTime2)*1000
    print("Waktu eksekusi brute force: {:.3f}".format(totalTime2), "ms")
    print("Jumlah operasi euclidean brute force: ", dc.counterEuclidean)
    print("Platform: ", platform.processor())
    print("Jarak terdekat: {:.3f}".format(d1))
    print("============================================================")
    print("Apakah anda yakin ingin melihat visualisasi? (y/n)")
    answer = input()
```

```
    if(answer == 'y'):
        if m == 1:
            vs.visualize_closest_pair1(points)
        elif m == 2:
            vs.visualize_closest_pair2(points)
        elif m == 3:
            vs.visualize_closest_pair3(points)
        else:
            print("Maaf, visualisasi hanya tersedia untuk 1, 2, dan 3
dimensi")
        print("Apakah anda ingin mengulang permainan? (y/n)")
        if input() == 'y':
            os.system('cls' if os.name == 'nt' else 'clear')
            main()
        else:

            print("Terima kasih telah menggunakan program kami!")
    else:
        print("Apakah anda ingin mengulang permainan? (y/n)")
        if input() == 'y':
            os.system('cls' if os.name == 'nt' else 'clear')
            main()
        else:

            print("Terima kasih telah menggunakan program kami!")

main()
```

BAB IV

HASIL

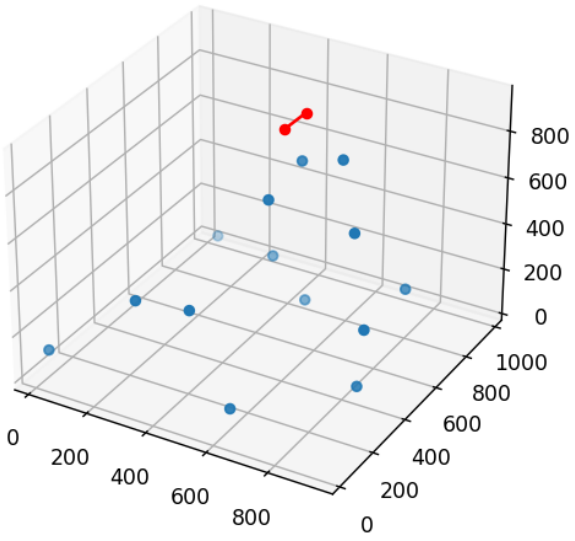
Program dijalankan pada spesifikasi komputer dengan sistem operasi Windows 11, processor AMD Ryzen 5, dan RAM 16GB. Dari spesifikasi tersebut menghasilkan hasil luaran program sebagai berikut :

A. Test Case n = 16

```
 _____    __      _____    _____    _____    _____    _____    _____    _____    __    _____
/\  __\  /\ \    /\  __ \  /\  __\  /\  __\  /\  __\  /\__  _\  /\  == \/\  __ \  /\ \  /\  == \
\ \ \___ \ \ \___ \ \ \/\ \ \ \___  \ \ \__   \ \___  \ \/_/\ \/  \ \  _-/\ \ \  __ \ \ \ \ \ \  __<
 \ \_____\ \ \_____\ \ \_____\ \/\_____\ \ \____\ \/\_____\   \ \_\   \ \_\  \ \_\ \_\ \ \_\ \ \_\ \_\
  \/_____/  \/_____/  \/_____/  \/_____/  \/____/  \/_____/    \/_/    \/_/   \/_/\/_/  \/_/  \/_/ /_/

Masukkan jumlah titik: 16
Masukkan jumlah dimensi: 3
```

```
============================================================
                Divide & Conquer Algorithm
============================================================
Waktu eksekusi divide and conquer: 1.002 ms
Jumlah operasi euclidean:  474
Jarak terdekat: 106.047
Platform:  AMD64 Family 23 Model 24 Stepping 1, AuthenticAMD
============================================================
                  Brute Force Algorithm
============================================================
Waktu eksekusi brute force: 0.000 ms
Jumlah operasi euclidean brute force:  599
Platform:  AMD64 Family 23 Model 24 Stepping 1, AuthenticAMD
Jarak terdekat: 106.047
============================================================
Apakah anda yakin ingin melihat visualisasi? (y/n)
```

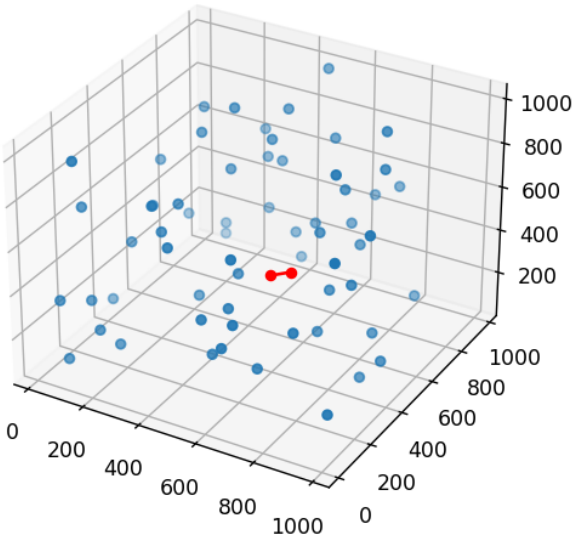


B. Test Case n = 64

```
 _____     __        _____     _____     _____     _____     _____     _____     _____     __    __     _____
/\  __-.  /\ \      /\  __ \   /\  __-.  /\  __-.  /\  __-.  /\__  _\   /\  == \ /\  __ \   /\ \  /\ \   /\  == \
\ \ \/\ \ \ \ \____ \ \ \/\ \  \ \ \/\ \ \ \ \/\ \ \ \ \/\ \ \/_/\ \/   \ \  _-/ \ \ \/\ \  \ \ \ \ \ \  \ \  __<
 \ \____-  \ \_____\ \ \_____\  \ \____-  \ \____-  \ \____-     \ \_\    \ \_\    \ \_____\  \ \_\ \ \_\  \ \_\ \_\
  \/____/   \/_____/  \/_____/   \/____/   \/____/   \/____/      \/_/     \/_/     \/_____/   \/_/  \/_/   \/_/ /_/

Masukkan jumlah titik: 64
Masukkan jumlah dimensi: 3
```

```
================================================================
                  Divide & Conquer Algorithm
================================================================
Waktu eksekusi divide and conquer: 3.989 ms
Jumlah operasi euclidean:  1724
Jarak terdekat: 97.000
Platform:  AMD64 Family 23 Model 24 Stepping 1, AuthenticAMD
================================================================
                    Brute Force Algorithm
================================================================
Waktu eksekusi brute force: 6.044 ms
Jumlah operasi euclidean brute force:  3753
Platform:  AMD64 Family 23 Model 24 Stepping 1, AuthenticAMD
Jarak terdekat: 97.000
================================================================
Apakah anda yakin ingin melihat visualisasi? (y/n)
```

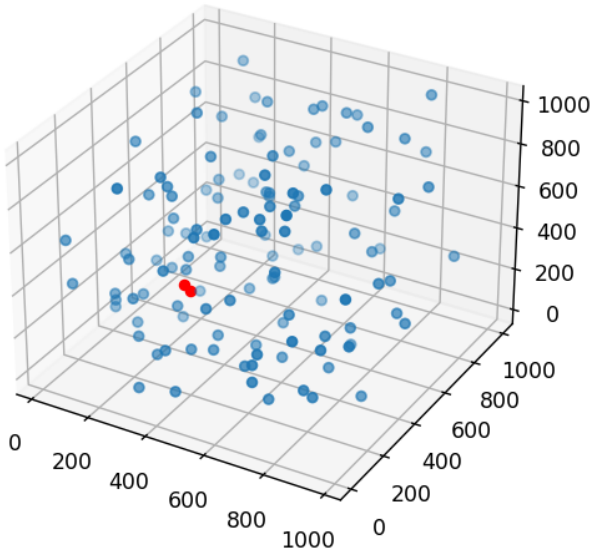


C. Test Case n = 128

```
 _____     __      _____   _____   _____   _____   _____    _____  _____    __    _____
/\  __-.  /\ \    /\  __-. /\  __-. /\  __-. /\  __-. /\__  _\  /\  == \ /\  __-. /\ \   /\  == \
\ \ \/\ \ \ \ \___ \ \ \/\ \ \ \___ \ \ \___ \ \ \___ \ \/_/\ \/  \ \  _-/ \ \ \/\ \ \ \ \  \ \  __<
 \ \____-  \ \_____\ \ \____-  \/\_____\ \ \____-  \/\_____\   \ \_\   \ \_\   \ \____-  \ \_\  \ \_\ \_\
  \/____/   \/_____/  \/____/   \/_____/  \/____/   \/_____/    \/_/    \/_/    \/____/   \/_/   \/_/ /_/

Masukkan jumlah titik: 128
Masukkan jumlah dimensi: 3
```

```
============================================================
                Divide & Conquer Algorithm
============================================================
Waktu eksekusi divide and conquer: 8.007 ms
Jumlah operasi euclidean:  6619
Jarak terdekat: 31.417
Platform:  AMD64 Family 23 Model 24 Stepping 1, AuthenticAMD
============================================================
                  Brute Force Algorithm
============================================================
Waktu eksekusi brute force: 42.996 ms
Jumlah operasi euclidean brute force:  14754
Platform:  AMD64 Family 23 Model 24 Stepping 1, AuthenticAMD
Jarak terdekat: 31.417
============================================================
Apakah anda yakin ingin melihat visualisasi? (y/n)
```



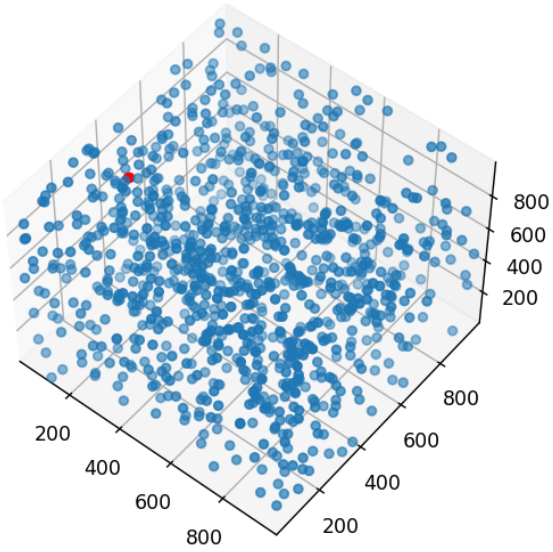
D. Test Case n = 1000

IF2211
Strategi Algoritma

```
 ______     __         ______     ______     ______     ______     ______     ______     __   __     ______
/\  ___\   /\ \       /\  __ \   /\  ___\   /\  ___\   /\  ___\   /\  ___\   /\  == \   /\ "-.\ \   /\  ___\
\ \ \____  \ \ \____  \ \ \/\ \  \ \___  \  \ \ \____  \ \___  \  \ \  __\   \ \  _-/   \ \ \-.  \  \ \ \____
 \ \_____\  \ \_____\  \ \_____\  \/\_____\  \ \_____\  \/\_____\  \ \_____\  \ \_\      \ \_\\"\_\  \ \_____\
  \/_____/   \/_____/   \/_____/   \/_____/   \/_____/   \/_____/   \/_____/   \/_/       \/_/ \/_/   \/_____/

Masukkan jumlah titik: 1000
Masukkan jumlah dimensi: 3
```

```
==========================================================
                Divide & Conquer Algorithm
==========================================================
Waktu eksekusi divide and conquer: 166.902 ms
Jumlah operasi euclidean:  47526
Jarak terdekat: 5.657
Platform:  AMD64 Family 23 Model 24 Stepping 1, AuthenticAMD
==========================================================
                  Brute Force Algorithm
==========================================================
Waktu eksekusi brute force: 1287.387 ms
Jumlah operasi euclidean brute force:  547037
Platform:  AMD64 Family 23 Model 24 Stepping 1, AuthenticAMD
Jarak terdekat: 5.657
==========================================================
Apakah anda yakin ingin melihat visualisasi? (y/n)
```



BAB V

KESIMPULAN

Algoritma divide and conquer dapat menyelesaikan permasalahan pencarian pasangan titik terdekat secara cepat. Hal ini dapat dibuktikan dengan perbandingan menggunakan algoritma brute force, ketika jumlah titik semakin banyak algoritma divide and conquer menghasilkan waktu eksekusi yang lebih cepat dan penggunaan rumus euclidean yang lebih sedikit dibandingkan dengan algoritma brute force.

IF2211
Strategi Algoritma

LAMPIRAN

Pranala Github : https://github.com/archmans/Tucil2_13521010_13521014.git

Checklist :

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	✔	