STRATEGI ALGORITMA
LAPORAN ANALISIS PERSOALAN KLASIK
(*CLOSEST PAIR*)

Ditujukan sebagai salah satu syarat
Untuk memperoleh nilai pada mata kuliah Strategi Algoritma
Program Studi DIV Teknik Informatika

Oleh :

FITRAH ALI AKBAR SETIAWAN (1214085)
RACHMA NURHALIZA PARINDRA (1214056)

UNIVERSITAS LOGISTIK DAN BISNIS INTERNASIONAL
PROGRAM STUDI DIV TEKNIK INFORMATIKA
BANDUNG
2023

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 *Closest Pair*

Closest Pair Point merupakan algoritma yang digunakan untuk mencari jarak terdekat antara kumpulan titik dalam suatu bidang dua dimensi [1]. Penentuan sensitivitas sensor warna pada pola tes (buta warna) dapat diuji dengan metode ini. Persamaan metode algoritma *closest pair point* sebagai berikut :

$$d = \sqrt{(Rd - Ri)\,2 + (Gd - Gi)\,2 + (Bd - Bi)\,2}$$

Penjelasaan dari persamaan diatas menyebutkan bahwa d sebagai nilai *closest pair point*, "Rd" sebagai nilai *red* warna objek, "Ri" sebagai nilai *red* pengukuran putih, "Gd" sebagai nilai *green* warna objek, "Gi" sebagai nilai *green* pengukuran putih, "Bd" sebagai nilai *blue* warna objek dan "Bi" sebagai nilai *blue* pengukuran putih [2].

1.2 Algoritma *Brute Force*

Algoritma *Brute Force* memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas. Namun, algoritma bisa sangat lambat dalam menyelesaikan masalah untuk beberapa kasus [3]. Secara konseptual, *Brute Force* bekerja sebagai berikut :

1. Mula-mula *pattern* dicocokkan pada awal teks.
2. Dengan bergerak dari kiri ke kanan, bandingkan setiap karakter di dalam *pattern* dengan karakter yang bersesuaian di dalam teks sampai :
    - Semua karakter yang dibandingkan cocok atau sama (pencarian berhasil), atau
    - Dijumpai sebuah ketidakcocokan karakter (pencarian belum berhasil)
3. Bila *pattern* belum ditemukan kecocokannya dan teks belum habis, geser *pattern* satu karakter ke kanan dan ulangi langkah 2 [4].

1.3 Algoritma *Divide and Conquer*

Divide and conquer adalah metode penyelesaian masalah dengan membagi masalah utama menjadi masalah yang lebih kecil. Pada strategi ini, terdapat 3 bagian utama dalam menyelesaikan suatu masalah yaitu *divide, conquer,* dan *combine.*

Divide yaitu membagi titik-titik itu ke dalam dua bagian, *PLeft* dan *PRight,* setiap bagian mempunyai jumlah titik yang sama. Sedangkan *conquer* yaitu secara rekursif, terapkan algoritma D-and-C pada masing - masing bagian [5].

BAB II
ANALISIS

2.1 Pengujian *Closest Pair Brute Force*

Berikut merupakan pengujian *Closest Pair* menggunakan algoritma *Brute Force*. Pada pengujian ini, penguji melakukan pengujian dengan menggukan *library pygame* untuk mengimplementasikan *Brute Force* pada permasalahan klasik *Closest Pair*. Penguji membuat beberapa titik secara acak dan mencari pasangan titik terdekat.

(Sumber : https://github.com/projeto-de-algoritmos/Divide-and-Conquer-List4-DanielGoncalves-LucasMacedo)

Source Code :

```
import sys
import pygame
import math

WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
YELLOW = (222, 178, 0)
PINK = (225, 96, 253)
BLUE = (0, 0, 255)
LIGHTORANGE = (255, 176, 56)
INTERMEDIARYORANGE = (255, 154, 0)
LIGHTBLUE = (60, 170, 255)
DARKBLUE = (0, 101, 178)
BEIGE = (178, 168, 152)

WIDTH = 950
HEIGHT = 650
SCREEN_SIZE = (WIDTH, HEIGHT)

def text_block(background, message, color, size, coordinate_x,
coordinate_y):
    """
    Create block of text on the screen.
    """
    font = pygame.font.SysFont(None, size)
    text = font.render(message, True, color)
    background.blit(text, [coordinate_x, coordinate_y])
```

```
def distance_two_points(point_a, point_b):
    """
    Calculate distance of two points.
    """
    distance = math.sqrt(pow((point_a.pos_x - point_b.pos_x), 2) +
pow((point_a.pos_y - point_b.pos_y), 2))
    return distance

def merge_sort_axis_x(points):
    """
    Sort points by axis x values.
    """
    if len(points) > 1:
        mid = len(points) // 2  # Finding the mid of the array
        L = points[:mid] # Divinding the array elements
        R = points[mid:] # into 2 halves

        merge_sort_axis_x(L) # Sorting the first half
        merge_sort_axis_x(R) # Sorting the second half

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i].pos_x < R[j].pos_x:
                points[k] = L[i]
                i += 1
            else:
                points[k] = R[j]
                j += 1
            k += 1

        # Checking if any elements was left
        while i < len(L):
            points[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            points[k] = R[j]
            j += 1
            k += 1


def merge_sort_axis_y(points):
    """
    Sort points by axis y values.
    """
    if len(points) > 1:
```

```
        mid = len(points) // 2 # Finding the mid of the array
        L = points[:mid] # Dividing the array elements
        R = points[mid:] # into 2 halves

        merge_sort_axis_y(L) # Sorting the first half
        merge_sort_axis_y(R) # Sorting the second half

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i].pos_y < R[j].pos_y:
                points[k] = L[i]
                i += 1
            else:
                points[k] = R[j]
                j += 1
            k += 1

        # Checking if any element was left
        while i < len(L):
            points[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            points[k] = R[j]
            j += 1
            k += 1

def closest_pair_of_points_brute_force(points):
    """
    Algorithm brute force to find closest pair of points.
    """
    min = sys.float_info.max
    for i in range(0, len(points)):
        for j in range(i + 1, len(points)):
            if distance_two_points(points[i], points[j]) < min:
                min = distance_two_points(points[i], points[j])
                closest_pair = (points[i], points[j])
    return closest_pair

def min_distance_strip(strip, closest_pair):
    """
    Calculate min distance in strip,
    strip is a vector of points in strip and
    closest_pair is current closest pair of points.
    """
```

```
    # Sort the vector of points by axis y
    merge_sort_axis_y(strip)
    i = 0
    while i < len(strip):
        j = i + 1
        # Check the distance to the following points whose distance is
less than closest pair
        while j < len(strip) and (strip[j].pos_y - strip[i].pos_y) <
distance_two_points(closest_pair[0], closest_pair[1]):
            # If distance minor that current closest pair, replace the
closest pair
            if (distance_two_points(strip[i], strip[j]) <
distance_two_points(closest_pair[0], closest_pair[1])):
                closest_pair = (strip[i], strip[j])
            j += 1
        i += 1

    # Return the closest pair of points between parameter and points of
strip
    return closest_pair

def closest_pair_of_points_divide_and_conquer(points):
    """
    Algorithm divide and conquer closest pair of points.
    """
    if len(points) <= 3:
        # Calculate by brute force when have less or equal three points
        return closest_pair_of_points_brute_force(points)

    mid = len(points) // 2 # Finding the mid of the array
    L = points[:mid] # Dividing the array elements
    R = points[mid:] # into 2 halves

    # Recursive call to left side
    dl = closest_pair_of_points_divide_and_conquer(L)
    # Recursive call to right side
    dr = closest_pair_of_points_divide_and_conquer(R)

    if distance_two_points(dl[0], dl[1]) <= distance_two_points(dr[0],
dr[1]):
        closest_pair = (dl[0], dl[1])
    else:
        closest_pair = (dr[0], dr[1])

    # Vector of points in strip
    strip = []
    # X axis value of central point
```

```
    mid_x = points[mid].pos_x
    i = 0
    while i < len(points):
        # Put on vector strip points between (mid_x + d) and (mid_x - d)
        if abs(points[i].pos_x - mid_x) <
distance_two_points(closest_pair[0], closest_pair[1]):
            strip.append(points[i])
        i += 1

    # Calculate minimum distance in strip
    mf = min_distance_strip(strip, closest_pair)
    if distance_two_points(closest_pair[0], closest_pair[1]) >
distance_two_points(mf[0], mf[1]):
        closest_pair = (mf[0], mf[1])

    # Return the closest pair of points
    return closest_pair


class Point():
    """
    Class to each point.
    """

    def __init__(self, pos, color):
        self.pos = pos
        self.pos_x = pos[0]
        self.pos_y = pos[1]
        self.color = color

    def render(self, background):
        """
        Render points on the screen.
        """
        pygame.draw.circle(background, self.color, self.pos, 8)


class Points():
    """
    Class to set of points.
    """

    def __init__(self):
        self.set_points = []
        self.set_points_positions = []

    def append_point(self, point):
        """
        Append created point to list of points.
```

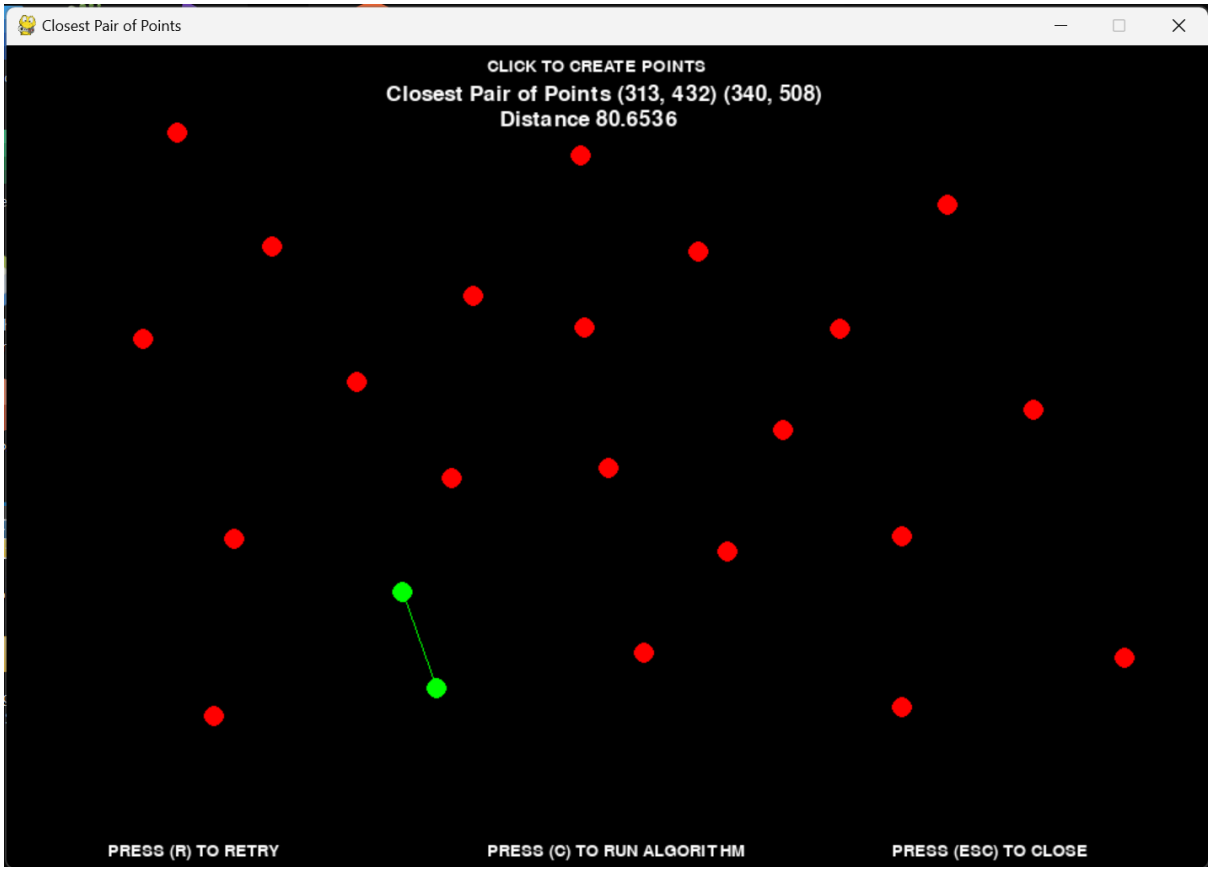
```
        """
        if ([point.pos_x, point.pos_y] not in self.set_points_positions):
            self.set_points.append(point)
            self.set_points_positions.append([point.pos_x, point.pos_y])

    def render(self, background):
        """
        Render list of points created.
        """
        for point in self.set_points:
            point.render(background)


class Game():
    """
    Class to manage the game.
    """

    def __init__(self):
        try:
            pygame.init()
        except:
            print('The pygame module did not start successfully')

        self.start = False
        self.exit = False
        self.solved = False
        self.closest_pair = ()

    def load(self):
        """
        Load necessary elements.
        """
        self.background = pygame.display.set_mode(SCREEN_SIZE)
        pygame.display.set_caption('Closest Pair of Points')

        self.points = Points()

    def initial_game(self):
        """
        Render home screen of program.
        """
        self.background.fill(DARKBLUE)
        pygame.draw.rect(self.background, BEIGE, [60, 60, 820, 520])
        pygame.draw.rect(self.background, LIGHTBLUE, [60, 120, 820, 400])
        pygame.draw.rect(self.background, BLACK, [130, 170, 680, 320])
        pygame.draw.rect(self.background, DARKBLUE, [130, 170, 680, 80])
```

```
        text_block(self.background, "CLOSEST PAIR OF POINTS",
LIGHTORANGE, 50, 250, 195)
        text_block(self.background, "PRESS (S) TO START",
INTERMEDIARYORANGE, 50, 290, 320)
        text_block(self.background, "PRESS (ESC) TO CLOSE",
INTERMEDIARYORANGE, 50, 270, 360)

    def founded(self):
        """
        Method to change color of closest pair of points.
        """
        self.closest_pair[0].color = GREEN
        self.closest_pair[1].color = GREEN

    def render(self):
        """
        Render elements on the screen.
        """
        self.background.fill(BLACK)

        self.points.render(self.background)

        text_block(self.background, "CLICK TO CREATE POINTS", WHITE, 20,
380, 10)
        text_block(self.background, "PRESS (R) TO RETRY", WHITE, 20, 80,
630)
        text_block(self.background, "PRESS (C) TO RUN ALGORITHM", WHITE,
20, 380, 630)
        text_block(self.background, "PRESS (ESC) TO CLOSE", WHITE, 20,
700, 630)

        if self.solved:
            pygame.draw.line(self.background, GREEN,
self.closest_pair[0].pos, self.closest_pair[1].pos)
            text_block(self.background, "Closest Pair of Points " +
str(self.closest_pair[0].pos) + " " + str(self.closest_pair[1].pos),
WHITE, 25, 300, 30)
            text_block(self.background, "Distance " +
str(round(distance_two_points(self.closest_pair[0],
self.closest_pair[1]), 4)), WHITE, 25, 390, 50)

        pygame.display.update()

    def run(self):
        """
        Method with loop to run game.
        """
        self.load()
```

```
        while not self.start:
            self.initial_game()

            if pygame.event.get(pygame.QUIT) or pygame.key.get_pressed()[
                    pygame.K_ESCAPE]:
                pygame.quit()
                sys.exit(0)
            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN and event.key ==
pygame.K_s:
                    self.start = True
                    self.background.fill(BLACK)

            pygame.display.update()

        while not self.exit:
            if pygame.event.get(pygame.QUIT) or pygame.key.get_pressed()[
                    pygame.K_ESCAPE]:
                self.exit = True
            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_r:
                        self.start = False
                        self.solved = False
                        self.run()
                    if event.key == pygame.K_c and
len(self.points.set_points) > 1:
                        if len(self.closest_pair) > 0:
                            self.closest_pair[0].color = RED
                            self.closest_pair[1].color = RED
                        merge_sort_axis_x(self.points.set_points)
                        #self.closest_pair =
closest_pair_of_points_divide_and_conquer(self.points.set_points)
                        self.closest_pair =
closest_pair_of_points_brute_force(self.points.set_points)
                        self.founded()
                        self.solved = True
                if event.type == pygame.MOUSEBUTTONUP:
                    pos = pygame.mouse.get_pos()
                    # Debug
                    print(pos)
                    point = Point(pos, RED)
                    self.points.append_point(point)
            self.render()

        pygame.quit()
        sys.exit(0)
```

```python
def main():
    """
    Main method.
    """
    mygame = Game()
    mygame.run()


if __name__ == '__main__':
    """
    Call main method.
    """
    try:
        main()
    except KeyboardInterrupt:
        print('Interruption')
```

Output :



Hasil yang didapat yaitu dari beberapa titik acak yang sudah dibuat penguji, titik (313, 432) memiliki jarak terdekat dengan titik (340, 508) dengan jarak 80.6536, dibandingkan dengan titik yang lain.

2.2 Pengujian *Closest Pair Divide and Conquer*

Berikut merupakan pengujian *Closest Pair* menggunakan algoritma *Divide and Conquer*. Pada pengujian ini titik-titik yang digunakan dalam pengujian yaitu (1, 2), (1, 3), (4, 6) dan (6, 7).

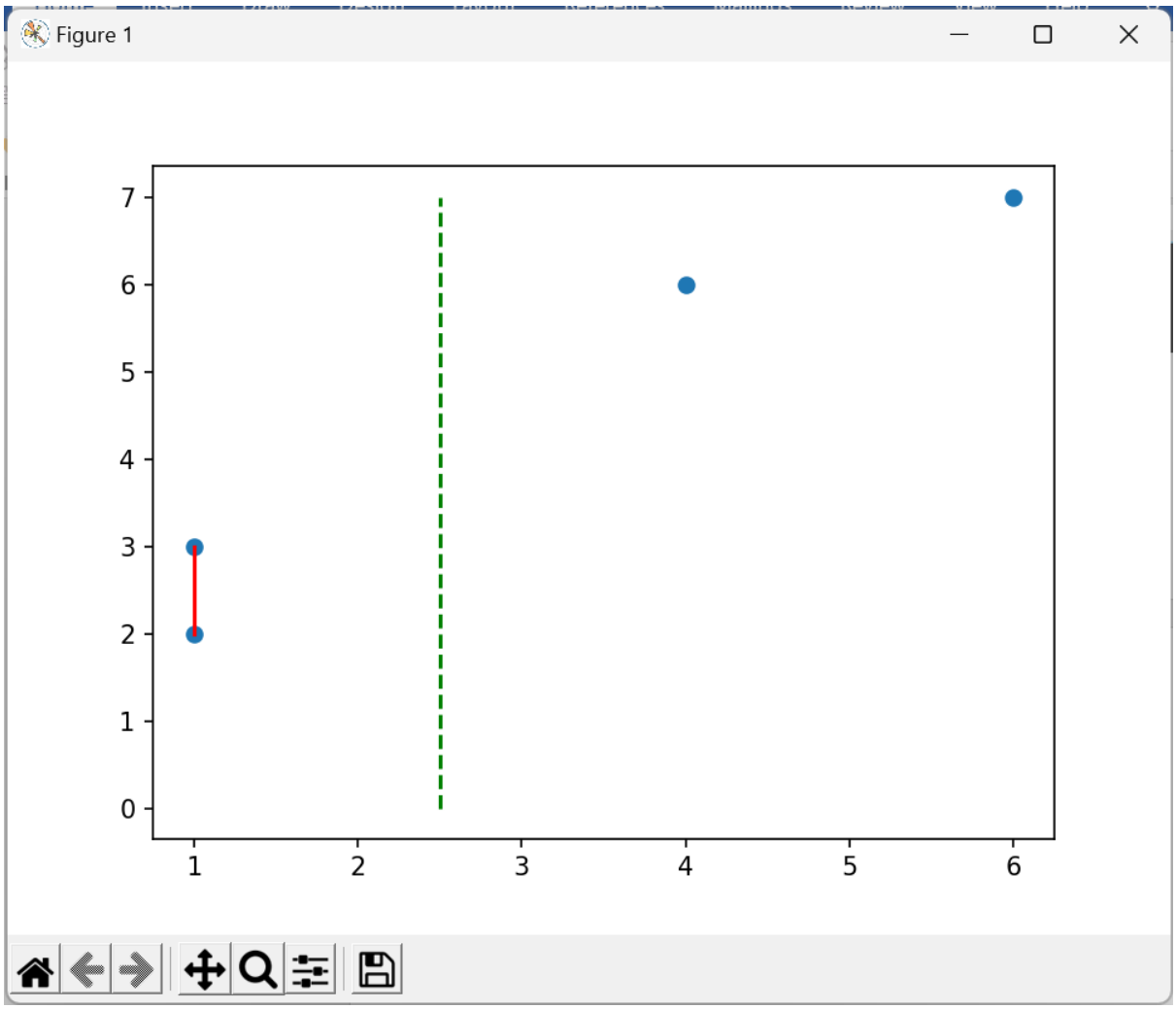
(Sumber : https://github.com/CosteanRobert/closestPairProblem)

Source Code :

```
from math import sqrt
import matplotlib.pyplot as plt

def minDistance(points):
    if len(points) < 2:
        return float('inf'), ()

    if len(points) == 2:
        dist = sqrt((points[0][0] - points[1][0]) ** 2 + (points[0][1] -
points[1][1]) ** 2)
        return dist, (points[0], points[1])

    points.sort(key=lambda p: p[0])

    n = len(points)
    if n % 2 == 0:
        median = (points[n // 2][0] + points[n // 2 - 1][0]) / 2
    else:
        median = points[n // 2][0]

    left_points = [p for p in points if p[0] <= median]
    right_points = [p for p in points if p[0] > median]

    min_left, pair_left = minDistance(left_points)
    min_right, pair_right = minDistance(right_points)

    delta = min(min_left, min_right)
    min_straddle, pair_straddle = straddleMin(points, median, delta)

    if min_left <= min(min_right, min_straddle):
        return min_left, pair_left
    elif min_right <= min(min_left, min_straddle):
        return min_right, pair_right
    else:
        return min_straddle, pair_straddle
```

```
def straddleMin(points, median, delta):
    points.sort(key=lambda p: p[1])
    minDist = float('inf')
    pair = ()
    n = len(points)
    for i, p1 in enumerate(points):
        for j, p2 in enumerate(points):
            if p1 == p2:
                continue
            if abs(p1[0] - p2[0]) > delta:
                continue
            dist = sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)
            if dist < minDist:
                minDist = dist
                pair = (p1, p2)
    return minDist, pair

def graphicalRepresentation(points, pair, median):
    xs, ys = zip(*points)
    plt.scatter(xs, ys)

    if pair:
        plt.plot([pair[0][0], pair[1][0]], [pair[0][1], pair[1][1]], 'r-
')
    plt.plot([median, median], [0, max(ys)], 'g--')
    plt.show()

def main():
    points = [(1, 2), (1, 3), (4, 6), (6, 7)]
    n = len(points)
    if n % 2 == 0:
        median = (points[n // 2][0] + points[n // 2 - 1][0]) / 2
    else:
        median = points[n // 2][0]
    minDist, closestPair = minDistance(points)
    print(f"Minimum distance: {minDist:.2f}")
    print(f"Closest pair: {closestPair}")
    graphicalRepresentation(points, closestPair, median)

if __name__ == '__main__':
    main()
```

Output :



Hasil yang didapat yaitu titik (1, 2) memiliki jarak terdekat dengan (1, 3) dibandingkan dengan (4, 6) dan (6, 7).

BAB III
PERBANDINGAN

3.1 Fitrah Ali Akbar Setiawan

Permasalahan closest pair adalah permasalahan yang jamak ditemukan dalam bidang matematika diskrit geometri. Permasalahan ini pada dasarnya hanya memiliki solusi dengan algoritma *brute force*. Meskipun demikian, terdapat sebuah rancangan solusi dengan algoritma *divide and conquer* yang memiliki kemangkusa lebih baik daripada algoritma *brute force*.

Meskipun demikian, pencapaian agoritma *divide and conquer* untuk persoalan ini tidak bersifat *straightforward*, melainkan memerlukan analisis mendalam terhadap sifat–sifat khusus yang dimiliki oleh permasalahan ini.

Oleh sebab itu, perolehan sebuah strategi algoritma tidak harus melalui algoritma–algoritma yang sudah ada saja. Pencapaian sebuah strategi sangat bergantung pada domain permasalahan yang dikaji. Untuk menemukannya, memang diperlukan pemahaman lebih terhadap domain permasalahan yang dihadapi.

Kesimpulannya, masalah *closest pair* adalah masalah klasik dalam ilmu komputer yang melibatkan pencarian dua titik dalam satu set titik dengan jarak terkecil di antara keduanya. Metode *brute force* memiliki kompleksitas waktu O(n^2) dan mudah diimplementasikan, tetapi mungkin tidak cocok untuk set poin yang besar. Metode *divide and conquer* memiliki kompleksitas waktu O(n log n) dan lebih efisien, tetapi membutuhkan kode yang lebih kompleks untuk diterapkan. Masalah *closest pair* memiliki banyak aplikasi praktis di berbagai bidang seperti pemrosesan gambar, grafik komputer, dan sistem informasi geografis.

3.2 Rachma Nurhaliza Parindra

Algoritma *divide and conquer* dan *algoritma burce force* memiliki persamaan yaitu membagi permasalahan utama menjadi beberapa permasalahan kecil. Tetapi ada beberapa perbedaan yang penting diperhatikan dari keduanya.

Divide and conquer adalah metode penyelesaian masalah dengan membagi masalah utama menjadi masalah yang lebih kecil. Pada strategi ini, terdapat 3 bagian utama dalam menyelesaikan suatu masalah yaitu *divide, conquer, dan combine. Divide* yaitu membagi titik-titik itu ke dalam dua bagian, *PLeft* dan *PRight*, setiap bagian mempunyai jumlah titik yang sama. Sedangkan *conquer* yaitu Secara rekursif, terapkan algoritma D-and-C pada masing - masing bagian.

Sedangkan, Algoritma *Brute Force* adalah teknik paling sederhana untuk menyelesaikan permasalahan komputasi pada umumnya. Secara konseptual, *Brute Force* bekerja sebagai berikut :

1. Mula-mula *pattern* dicocokkan pada awal teks.
2. Dengan bergerak dari kiri ke kanan, bandingkan setiap karakter di dalam *pattern* dengan karakter yang bersesuaian di dalam teks sampai :
    - Semua karakter yang dibandingkan cocok atau sama (pencarian berhasil), atau
    - Dijumpai sebuah ketidakcocokan karakter (pencarian belum berhasil)
3. Bila *pattern* belum ditemukan kecocokannya dan teks belum habis, geser *pattern* satu karakter ke kanan dan ulangi langkah 2.

Jadi, Dapat disimpulkan bahwa strategi *Divide and Conquer* lebih efektif dan efisien dalam menangani masalah pengurutan. Algoritma yang mudah dalam hal implementasi adalah *Bubble Sort* dan *Selection Sort*, keduanya memiliki kompleksitas O (n2). Algoritma yang lebih efisien adalah algoritma *Quick Sort* dan *Merge Sort* dengan kompleksitasnya adalah O (n log n).

DAFTAR PUSTAKA

[1] M. a. S. R. Yusuf, ”RANCANG BANGUN PROSES SORTIR BENDA BERDASARKAN WARNA MENGGUNAKAN ALGORITMA CLOSED PAIR POINT DAN SENSOR TCS3200,” *SEMINAR TEKNOLOGI TERAPAN* , Vol. %1 av %2Vol. 1, No. 1, pp. 108-114, 2021.

[2] D. M. Dzulkiflih, ”UJI SENSITIVITAS SENSOR TCS230 BERBASIS ARDUINO UNO SEBAGAI ALAT PENDETEKSI WARNA BAGI PENDERITA BUTA WARNA,” *Jurnal Inovasi Fisika Indonesia (IFI),* vol. 10, pp. 43-51, 2021.

[3] I. G. W. Kusuma Jaya, I. B. N. W. Manuaba, K. R. Wijaya, I. P. S. Pratama Wardhana, I. M. A. Saputra och I. G. A. Gunadi, ”Analisis Komparasi Algoritma Sorting Antara Metode Brute Force dengan Divide and Conquer,” *Jurnal Ilmu Komputer Indonesia(JIK),* vol. 5, pp. 2615-2703, 2020.

[4] F. A. T. Tobing och J. R. Tambunan, ”Analisis Perbandingan Efisiensi Algoritma Brute Force dan Divide and Conquer dalam Proses Pengurutan Angka,” vol. XII, pp. 57-58, 2020.

[5] P. och D. Yudi, ”Pemanfaatan GPS Sebagai Sarana Mendapatkan Pertolongan Ketika dalam Kondisi Bahaya dengan Algoritma Divide and Conquer untuk Menentukan Lokasi Terdekat,” *Jurnak Teknik Informatika (JIKA),* pp. 30-33, 2019.

[6] D. Goncavales och L. Macedo, ”GitHub,” 7 June 2019. [Online]. Available: https://github.com/projeto-de-algoritmos/Divide-and-Conquer-List4-DanielGoncalves-LucasMacedo. [Använd 7 March 2023].

[7] ”GitHub,” 25 December 2022. [Online]. Available: https://github.com/CosteanRobert/closestPairProblem. [Använd 7 March 2023].