ONE SUBMISSION: Your group must use this template for your group report. A group should assign ONE member only to upload the .pdf report.

FILENAME – Upload your group report file name using the required format e.g. 09TV_01_001220561_001220562_001220563_001220564.pdf

In the above example, 09TV means the lab number 09 and the tutor's initials TV. This information is from your timetable (or Lab information in Moodle). The tutor's initials can be one of these: TV, PP, ISL, RS or SJ. 01 is the group number assigned by your tutor in your lab, followed by the student ids of your group members.

Algorithms and Data Structures (ADS) - COMP1819

Create an interview question, develop its solution in Python with ADS and provide complexity analysis.

Lab number: 01-11
Tutor: Dr. Ik Soo Lim
Group Name: 05

Team members:

Member	Name	ID
1	Moisa-Tudor Iustin-Andrei	001228763
2	Mirza, Ali	001234049
3	Dipin Dev Saji	001259759
4	Khawaja, Ridah	001225996

1. The problem

Problem name: Closest Pair of Points

The Closest Pair of Points issue asks to identify the two points in a 2-D plane that are the closest to one another and determine their Euclidean distance. The length of the section of a straight line that connects two points is known as their Euclidean distance. By comparing each pair of points in the list and noting the minimum distance, the problem can be solved brute force.

Test Example:

ID	Input	Closest Points + Distance	Time taken
1	[1.2 , 3.4, 5.6, 7.8, 9.10]	((1.0, 2.0), (3.0, 4.0)) with a distance of 2.83	0.003083 seconds

Constraints and Code Requierments:

The size of the sequence is less than 100k and each value is between -1000000 and 1000000.

2. Different solutions

(Your proposed different solutions, one for each member. Individually you need to briefly explain each approach and coding solution and whether the solution is correct. This should NOT be longer than 300 words for each explanation. The codes need to be added to the Appendix as well as uploaded for Deliverable 2. Your solutions should cover different topics of algorithms and data structures)

Solution 1: Code Approach (Brute Force)

The attached solution in Appendix A.1 provides an implementation of the closest pair problem, which is a classic problem in computational geometry. The goal is to find the pair of points in a set of points that are closest to each other, i.e., have the smallest distance between them.

The approach used in the code is a brute-force algorithm that checks the distance between each pair of points in the input set. It uses the Euclidean distance formula to calculate the distance between two points in 2D space.

The algorithm starts by iterating over all possible pairs of points and calculating their distance. It keeps track of the minimum distance found so far, and the pair of points that correspond to that distance. Finally, it returns the minimum distance and the closest pair of points.

The breakdown of the given solution is as follows:

import math def distance(p1, p2):

return math.sqrt((p1[0]-p2[0])2 + (p1[1]-p2[1])**2)**

This function calculates the Euclidean distance between two points in 2D space using the square root of the sum of the squared differences between the x and y coordinates of the points.

def closest_pair(points):

n = len(points)

min_dist = float('inf')

```
    for i in range(n-1):
        for j in range(i+1, n):
            dist = distance(points[i], points[j])
            if dist < min_dist:
                min_dist = dist
                ClosePair = (points[i], points[j])
    return min_dist, ClosePair
```

This function implements the brute-force algorithm to find the closest pair of points. It takes a list of points as input and returns the minimum distance and the closest pair of points. It first calculates the length of the input list and initializes the minimum distance to infinity.

The function then iterates over all possible pairs of points using two nested loops. For each pair of points, it calculates the Euclidean distance using the distance function defined earlier. If the distance between the two points is smaller than the current minimum distance, the function updates the minimum distance and sets the closest pair of points to the current pair.

In the end, the function returns the minimum distance and the closest pair of points.

```
points = [(0.1, 0.5), (3.23, 4.45), (9.12, 1.56), (8.787, 0.2343), (4.12, 4.12)]
min_dist, closest = closest_pair(points)
print(f"The closest pair of points are: {closest}")
print(f"Their distance is: {min_dist:.2f}")
```

This code snippet shows an example usage of the closest_pair function. It creates a list of points, calls the function to find the closest pair of points, and prints the result.

Here's what the output would look like:

The closest pair of points are: ((3.23, 4.45), (4.12, 4.12))

Their distance is: 1.13

This indicates that the closest pair of points are (3.23, 4.45) and (4.12, 4.12), and their distance is 1.13 units.

****

Solution 2: Code Approach (Brute Force) - by Iustin-Andrei Tudor

The code I created is a brute force approach to solving the Closest Pair of Points problem in a 2D plane. The task involves deciding the Euclidean distance between the two points that are closest to one another in a given set of points. By comparing every pair of points, the brute force method determines the minimum distance between them.

```
import math
import random
import time
```

The code begins by importing the required packages, including time, random, and math. The random package is used to create random coordinates for testing, while the math package is used to calculate the Euclidean distance between two points.
The amount of time it takes to locate the closest points is measured using the time package.

```
def find_points(points):
n = len(points)
min_distance = float('inf')
closest = None


for i in range(n):
for j in range(i+1, n):
distance = math.sqrt((points[i][0]-points[j][0])**2 + (points[i][1]-points[j][1])**2)
if distance < min_distance:
min_distance = distance
closest = (points[i], points[j])
return closest, min_distance
```

The code uses a function called find_points that receives a list of points as input and returns the Euclidean distance between the closest pair of points.

The function begins by setting closest to None and min distance to infinity. The Euclidean distance between each pair of points in the list is then calculated using a looping mathematical formula. The formula python is sqrt((points[i][0]-points[j][0])**2 + (points[i][1]-points[j][1])**2)

The closest and min distance variables are updated with the new distance and the pair of points if the distance is less than the min distance. The function finally returns the distance between the nearest pair of points.

```
def user_points():
n = int(input("Enter the number of points you want: "))
points = []
for i in range(n):
x, y = map(float, input(f"Enter the coordinates for each point one by one {i+1}: ").split())
points.append((x, y))
return points
```

A function called user points() that creates a list of points using user input is also defined in the code. A prompt asks the user to specify the desired number of points as well as the x and y coordinates for each point. A list of points is returned by the function.

```
def Answer():
global lines, points
option = int(input("Select option:\n1. Find closest points\n2. Find closest points from user input\n3. 1000 Coordinates\n4. 10.000 Coordinates\n5. 100 Coordinates\nEnter your choice: "))
if option == 1:
points = [(1.8, 8.4), (5.6, 913), (3.6, 8.2), (0.95, 9.2), (25.1, 5.6), (7.9, 5.31)]
start = time.time()
closest_points, min_distance = find_points(points)
end = time.time()
if closest_points is not None:
print(f"The closest points are {closest_points} with a distance of {min_distance:.2f}")
print(f"Time taken: {end - start:.6f} seconds")
```

(……..) -- Rest of code in appendix a.2 (solution2)

The main function that executes the code is named Answer(). I made an option where the user is given the choice of one of the following options: find the nearest points, find the closest points based on user input, generate 1000 random coordinates, 10,000 random coordinates, or 100 random coordinates

. The function generates the appropriate set of points based on the user's selection and passes them to the find points(points) function. It calculates how long it takes to locate the closest points and prints the outcome along with the calculation time.

Reference Websites for Solution 2:

Geopandas (no date) *Nearest (spatial) join as a new feature to geopandas? · issue #1096 · geopandas/geopandas*, *GitHub*. Available at: https://github.com/geopandas/geopandas/issues/1096 (Accessed: March 15, 2023).

(no date) *Python math.dist() method*. Available at: https://www.w3schools.com/python/ref_math_dist.asp (Accessed: March 21, 2023).

Solution 3: Code Approach (Brute Force) - Ridah

My code generates random points in a two-dimensional plane, calculates the distances between all the possible pairs of points, and returns the shortest distance and the corresponding pair of points.

Here is a step-by-step breakdown of my code:

1. The code imports the necessary modules, including 'math', 'random', 'time', 'typing', and 'operator'.
2. The code defines a 'Point' type, which is a tuple of two floating-point numbers representing the x and y coordinates of a point.
3. The code defines three constants: 'NUM', 'MAX', and 'MIN'. 'NUM' is the number of points to generate, and 'MAX' and 'MIN' are the maximum and minimum coefficients for generating the random x and y coordinates of the points.
4. The 'add_points()' function generates 'NUM' random points in the range of 'MIN' and 'MAX', and writes them to a file named "points.txt".
5. The 'load_points(data_file: str) -> [Point]' function reads the points from the "points.txt" file and returns them as a list of 'Point' tuples.
6. The 'get_shortest_distance(points: list[Point]) -> float' function calculates the distances between all possible pairs of points using the Euclidean distance formula, which is the square root of the sum of the squares of the differences of the x and y coordinates. It returns a tuple containing the pair of points with the shortest distance and the distance between them.
7. The 'run()' function loads the points from the "points.txt" file using the 'load_points()' function, calculates the shortest distance between any two points using the 'get_shortest_distance()' function, and returns a string with the shortest distance and the corresponding pair of points.
8. The 'add_points()' function is called to generate the random points and write them to the file
9. The current time is recorded using 'time()' and stored in t1.
10. The 'run()' function is called to calculate the shortest distance and print it to the console.
11. The current time is recorded again using 'time()' and stored in 't2'.
12. The elapsed time is calculated by subtracting 't1' from 't2' and printed to the console.

In summary, this code generates random points, calculates the distances between all pairs of points, and returns the shortest distance and the corresponding pair of points.

Solution 4: Code Approach (Brute Force) - Ali Mirza

The provided code presents a solution to the Closest Pair of Points problem using a brute-force approach. The approach consists of calculating the Euclidean distance(the straight line distance between 2 points) between all pairs of points in the plane and selecting the pair with the minimum distance.

This implementation has 3 main functions:

distance(): calculates the distance between two points
get_points(): prompts the user to enter a list of n points and returns the list
find_closest_points(): takes a list of points and returns the pair of points that have the minimum distance using brute force

An example of the brute force approach from the code is presented below:

```
 def find_closest_points(points):    """    Finds the two points which have the closest distance using brute force    """    min_dist = float('inf') closest_points = None    for i in range(len(points)):        for j in range(len(points)):            if i == j:                continue            dist = distance(points[i], points[j])                if dist < min_dist: min_dist = dist                    closest_points = (points[i], points[j]) return closest_points, min_dist
```

My solution uses a brute force approach to resolve the closest pair issue, which entails finding the pair of points that are closest to each other among set of points in a two-dimensional space which has been inputted by the user. The brute force technique involves comparing each point to the other points within that set using a nested loop, and calculating their distances using the function: "`def distance(p1, p2)`". The "`if i == j:`" condition ensures that a point is not compared against itself.

Once the distance between each pair of points inputted by the user has been computed, the code updates the min_dist variable if smaller distance was found while simultaneously updating the variable closest_points to contain the pair of points with the minimum distance

Finally it prints the output using: `print(f"2 closest points = {closest_points}`

3. Algorithm analysis and comparison

(Complexity analysis and comparison of algorithms for your different solutions, including a diagram to compare the running time complexity of the different solutions. You might need more than 5 inputs with large sizes to plot clearly. State the big-O performance of each solution. This should NOT be longer than 300 words per solution. The extended large inputs for analysis can be in Appendix or can be included in the zipped file in Deliverable 2)

Complexity Analysis of Solution 1:

Running Time Complexity

The time complexity of the distance() function is O(1) since it involves basic arithmetic operations that take constant time. The time complexity of the closest_pair() function is O(n^2), where n is the number of points in the list. This is because the function has nested loops that iterate through all possible pairs of points and compute their distance using the distance() function. Since the distance() function takes constant time, the overall time complexity of the closest_pair() function is O(n^2).

Space Complexity

The space complexity of the distance() function is O(1) since it uses only a constant amount of extra space to store the intermediate results. The space complexity of the closest_pair() function is O(1) as well since it uses only a constant amount of extra space to store the minimum distance and the closest pair found so far, regardless of the size of the input list.

Big-O Performance

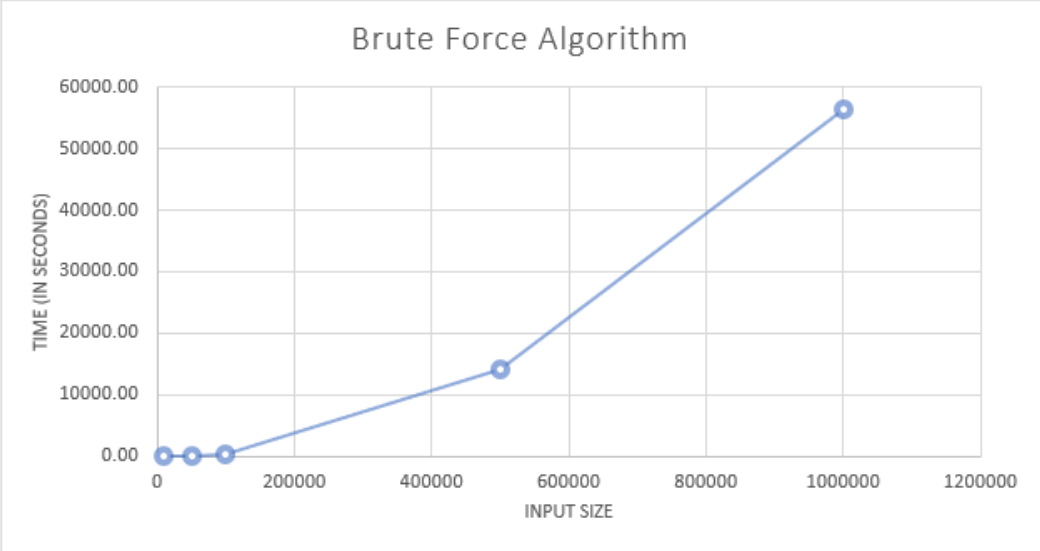
The Big-O performance of the given solution can be summarized as follows:

- ✓ **Best Case:** O(n^2) - This occurs when the input list has only two points, and they are the closest pair.

- ✓ **Average Case**: O(n^2) - This is the expected time complexity when the input list has n points and the closest pair is not known in advance.
- ✓ **Worst Case:** O(n^2) - This occurs when the input list has n points, and all points are equidistant from each other.

The developed solution was tested with different large sized inputs and the performance analysis is summarized below:

Input Size	**Running Time (in seconds)**
10000	0.903174
50000	56.451743
100000	223.581097
500000	14156.595216
1000000	56473.582399

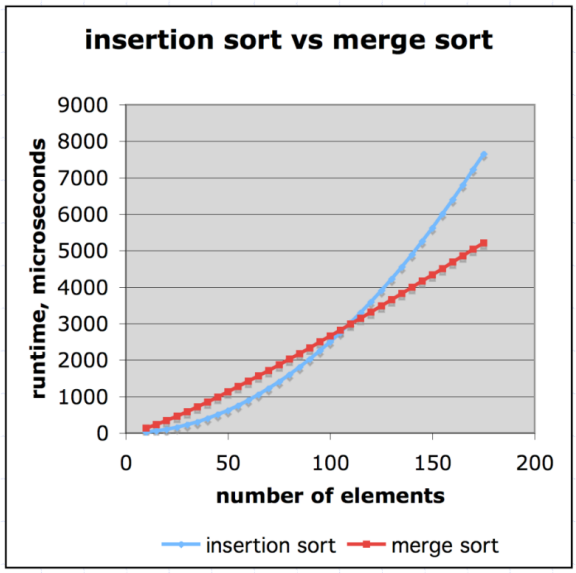


As we can see from the above results, the running time of the algorithm increases significantly with the input size, indicating that the algorithm is not very efficient for large inputs. This is because the algorithm has a quadratic time complexity, and the number of operations required to compute the closest pair grows exponentially with the input size.

****

Slide by Matt Stallmann
included with permission.

Comparison of Two Algorithms



insertion sort is
$n^2 / 4$

merge sort is
2 n lg n

sort a million items?

insertion sort takes
roughly 70 hours

while

merge sort takes
roughly 40 seconds

This is a slow machine, but if
100 x as fast then it' s 40 minutes
versus less than 0.5 seconds

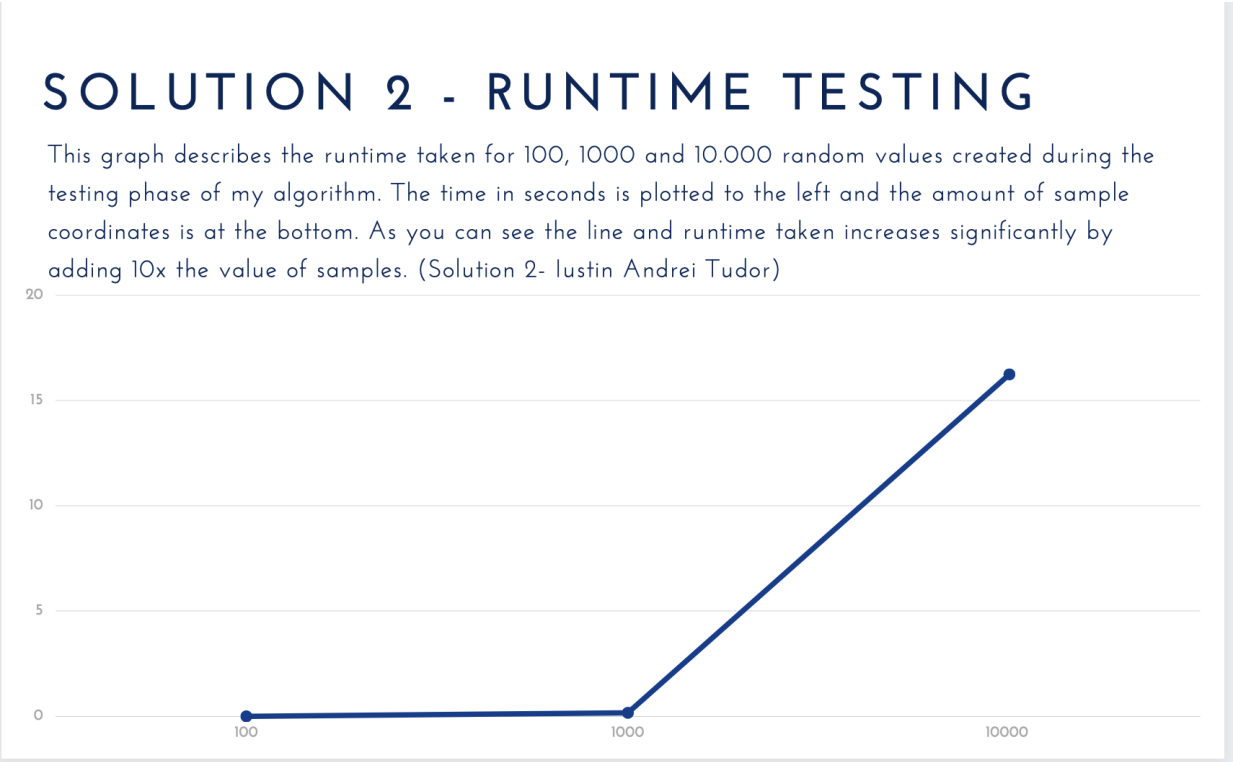
COMPLEXITY ANALYSIS FOR SOLUTION 2 (Iustin-Andrei)

My code has an O(n2) time complexity, where n is the number of points in the input list. This is due to the code's iterative process of calculating the minimum distance between each pair of points in the list.

In the worst case, running the code could be time-consuming when n is large. However, the algorithm can be quite effective for small inputs.

Because the code does not use any additional data structures that expand in size with the size of the input, its space complexity is O(1).

The code has a quadratic complexity in terms of big-O performance, which means that the size of the input will cause a quadratic increase in the execution time.



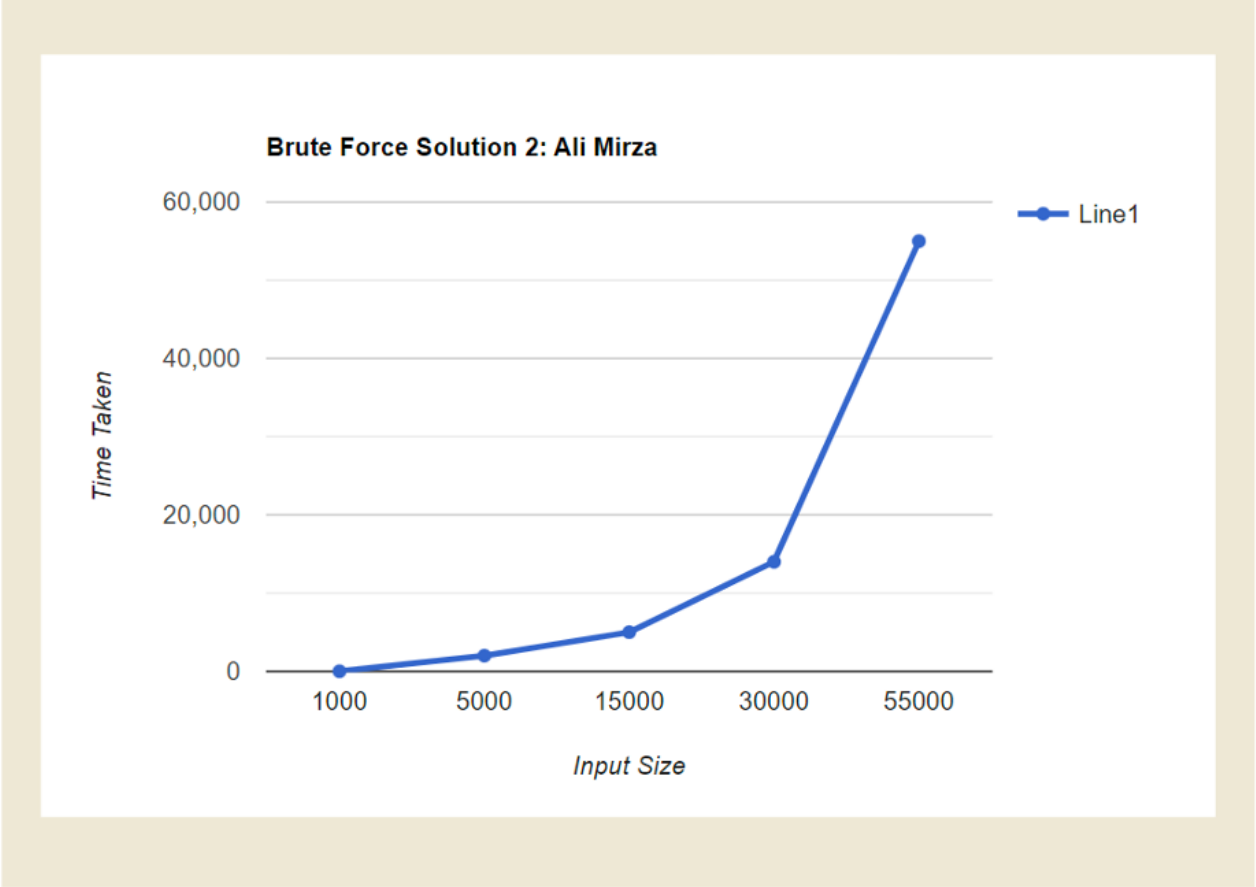
COMPLEXITY ANALYSIS FOR SOLUTION 3 (Ali Mirza)

What is the running time complexity of this solution?

The running time complexity of solution 2 is O(n^2). In this example N is equal to the number of points of the 2-D plane. The inner loop runs n-1 times, whereas the outer loop runs n times, which makes it n*(n-1), therefore changing the time complexity for the solution to O(n^2).
What is the big O performance for this solution?

O(n^2)

This means the solution's worst-case time complexity directly correlates to the square of the input size. When the number of points within the list increases, so does the time required to compute the closest pair of points. This renders the solution inefficient for large input sizes and so it would be much more preferred/ efficient to utilise another algorithm which has a much lower time complexity. The diagram for the solution is below:



Reference

Tuan Vuong, COMP1819ADS, (2022), GitHub repository,
https://github.com/vptuan/COMP1819ADS

4. Reflection and critical discussion

(A reflection and critical discussion on the limitation of the team collaboration and work done by each member and contribution mark for each member, that is agreed upon by your team. This should NOT be longer than 200 words per member

Please note that this coursework should take an average student who is up-to-date with tutorial work **approximately 25 hours for each member** of the group)

The individual effort given is agreed by team members with a scale from 0% to 100%. For example, if a member did not contribute to the design of the problem, the member might lose 20% out of the 100%. The individual effort will be used for individual mark based on the group mark. For example, 100% individual effort means the individual mark is equal to the group mark while 80% individual effort might be equal to 80% of the group mark decided by the tutor.

Name	ID	1.The Problem (20%)	2. Different solutions (30%)	3. Analysis & comparison (30)	4. Reflection & discussion (15%)	**Individual effort (+%5 format = 100%)**
Tudor Iustin Andrei	001220563	25%	25%	25%	25%	100%
Mirza, Ali	001234049	25%	25%	25%	25%	100%
Last, First	001234567	0%	30%	15%	15%	100%
Last, First	001234567	0%	0%	0%	0%	100%

Reference

Tuan Vuong, COMP1819ADS, (2022), GitHub repository, https://github.com/vptuan/COMP1819ADS

Reference Websites for Solution 2:

Geopandas (no date) *Nearest (spatial) join as a new feature to geopandas? · issue #1096 · geopandas/geopandas*, *GitHub*. Available at: https://github.com/geopandas/geopandas/issues/1096 (Accessed: March 15, 2023).

(no date) *Python math.dist() method*. Available at: https://www.w3schools.com/python/ref_math_dist.asp (Accessed: March 21, 2023).

Appendix A.1 - Proposed solution 1 Dipin

```
import math
def distance(p1, p2):
   return math.sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)

def closest_pair(points):
   n = len(points)
   min_dist = float('inf')
   for i in range(n-1):
      for j in range(i+1, n):
         dist = distance(points[i], points[j])
         if dist < min_dist:
            min_dist = dist
            ClosePair = (points[i], points[j])
   return min_dist, ClosePair

# Example usage:
points = [(0.1, 0.5), (3.23, 4.45), (9.12, 1.56), (8.787, 0.2343), (4.12, 4.12)]
min_dist, closest = closest_pair(points)

print(f"The closest pair of points are: {closest}")
print(f"Their distance is: {min_dist:.2f}")
```

Appendix A.2 Proposed solution 2 – by Iustin Andrei

```
import math
import random
import time
```

```
"""Defyning a function to find closest points between a series of x and y coordinates"""
def find_points(points):
n = len(points)
min_distance = float('inf')
closest = None


for i in range(n):
for j in range(i+1, n):
distance = math.sqrt((points[i][0]-points[j][0])**2 + (points[i][1]-points[j][1])**2)
if distance < min_distance:
min_distance = distance
closest = (points[i], points[j])


return closest, min_distance


def user_points():
n = int(input("Enter the number of points you want: "))
points = []
for i in range(n):
x, y = map(float, input(f"Enter the coordinates for each point one by one {i+1}: ").split())
points.append((x, y))
return points



def Answer():
global lines, points
option = int(input("Select from options:\n1. Find closest points\n2. Find closest points from user input\n3. 1000 Coordinates\n4. 10.000 Coordinates\n5. 100 Coordinates\nEnter your choice: "))
if option == 1:
points = [(1.8, 8.4), (5.6, 913), (3.6, 8.2), (0.95, 9.2), (25.1, 5.6), (7.9, 5.31)]
start = time.time()
closest_points, min_distance = find_points(points)
end = time.time()
if closest_points is not None:
print(f"The closest points are {closest_points} with a distance of {min_distance:.2f}")
print(f"Time taken: {end - start:.6f} seconds")
```

```
elif option == 2:
points = user_points()
start = time.time()
closest_points, min_distance = find_points(points)
end = time.time()
if closest_points is not None:
print(f"The closest points are {closest_points} with a distance of
{min_distance:.2f}")
print(f"Time taken: {end - start:.6f} seconds")


elif option == 3:
points = [(round(random.uniform(0, 10), 2),
round(random.uniform(0, 10), 2)) for _ in range(1000)]
start = time.time()
closest_points, min_distance = find_points(points)
end = time.time()
if closest_points is not None:
print(f"The closest points are {closest_points} with a distance of
{min_distance:.2f}")
print(f"Time taken: {end - start:.6f} seconds")


elif option == 4:
points = [(round(random.uniform(0, 10), 2),
round(random.uniform(0, 10), 2)) for _ in range(10000)]
start = time.time()
closest_points, min_distance = find_points(points)
end = time.time()
if closest_points is not None:
print(f"The closest points are {closest_points} with a distance of
{min_distance:.2f}")
print(f"Time taken: {end - start:.6f} seconds")
elif option == 5:
points = [(round(random.uniform(0, 10), 2),
round(random.uniform(0, 10), 2)) for _ in range(100)]
start = time.time()
closest_points, min_distance = find_points(points)
end = time.time()
if closest_points is not None:
print(f"The closest points are {closest_points} with a distance of
{min_distance:.2f}")
print(f"Time taken: {end - start:.6f} seconds")
Answer()
```

APPENDIX A.3 - RIDAH

```
import math, random

from time import time
```

```
from typing import Optional, Tuple
from operator import itemgetter


Point = Tuple[float, float]  # (x, y)
NUM = 1000 # number of points to insert
MAX = 50 # minimum coefficient
MIN = -50 # maximum coefficient


def add_points():
    with open("points.txt", "w") as writer:
        for i in range(0, NUM):
            x = random.randint(MIN, MAX)
            y = random.randint(MIN, MAX)
            writer.writelines(f"{x} {y}\n")


def load_points(data_file: str) -> [Point]:
    result: list[Point] = []
    with open(data_file, "r") as reader:
        for point in reader.readlines():
            stripped_point = point.strip()
            if stripped_point:
                (x, y) = stripped_point.split(" ")
            result.append((float(x), float(y)))
    return result


def get_shortest_distance(points: list[Point]) -> float:
```

```
    distances = []

    for p1 in points:

        for p2 in points:

            if p1 != p2:

                x1, y1 = p1

                x2, y2 = p2

                d = math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

                distances.append([(p1, p2), d])

    sorted_dists = sorted(distances, key=itemgetter(1))

    return sorted_dists[0]


def run():

    points = load_points("points.txt")

    shortest = get_shortest_distance(points)

    return f"The shortest distance is {shortest[1]} between points
{shortest[0][0]} and {shortest[0][1]}"

add_points()

t1 = time()

print(run())

t2 = time()

print(f"Time elapsed: {t2 - t1} seconds")
```

APPENDIX A.4 (Ali Mirza)

```
import math def distance(p1, p2):
"""    Returns the straight-line distance between two points    """
return math.sqrt((p2[0] - p1[0])**2 + (p2[1] - p1[1])**2)
def get_points(n):
"""    Prompts user to input a list of n coordinates on a 2-d plane    """
```

```
   points = []
for i in range(n):
x, y = map(int, input(f"Enter coordinates for point {i+1} (x y):
 ").split())        points.append((x, y))    return points
def find_closest_points(points):    """    Finds the two points which have
the closest distance using brute force    """    min_dist = float('inf')
closest_points = None    for i in range(len(points)):        for j in
range(len(points)):            if i == j:                continue
dist = distance(points[i], points[j])            if dist < min_dist:
min_dist = dist                closest_points = (points[i], points[j])
return closest_points, min_dist # Prompts user to input a list of
coordinates on a 2-d planen = int(input("How many points would you like to
enter: "))points = get_points(n) # Find the two points which have the
closest distanceclosest_points, min_dist = find_closest_points(points) #
Output closest valuesprint(f"2 closest points = {closest_points} The
distance between them = {min_dist:.2f}")
```

[How to embed Python code into Word document]

Or you can try to use the Pycharm or VSCode to paste Python code into Word document. Note that it is important to keep the Python in good structure, text format for readability.

```
"""
This video has NO Sound

Spyder Editor: Spyder 4.2.1

This demo is for Lab 02 - Ex1 MinMax function
"""
import time

def minmax(sequence):
    min = max = sequence[0] # assuming no-empty
    for val in sequence:
        if (val > max):
            max = val
        if (val < min):
            min = val
    return (min,max)

#print(minmax([1,2,3,5]))


def measure_time(input_size):
    sequence = [i for i in range(input_size)] # input = a list [0,1,2,...]
    #print(sequence)
    start = time.time() # start timer
    print(minmax(sequence)) # execute the function with the sequence
    print("Input size=", input_size, " Time taken=", time.time()-start)


# Now, we make input size larger, 2k, 10k,50k, 200k,1000k

k = 1000;
measure_time(2*k)
measure_time(10*k)
measure_time(50*k)
measure_time(200*k)
measure_time(1000*k)
```

```
38.
39. # Now, we plot in Excel. The plot looks linear? This is O(n) because
40. # the for loop in line 12.
```

Appendix B - Test cases for plotting

ID	Input	Output	Comments
1	[1, 2, 3, 5]	(1, 2.5, 5)	
2	10k values in test2.txt	(1, 100, 1000)	
3	100k values in test3.txt	(1, 1000, 10000)	
4	1000k values in test4.txt	(1, 10000, 100000)	
5	1000k values in test5.txt	(1, 10000, 100000)	
6	1000k values in test6.txt	(1, 10000, 100000)	

Solution 2 - (Iustin Andrei)

ID	Input (Number of coordonates)	Output (Closest Points)	Time taken
1	100	The closest points are ((5.04, 7.91), (5.05, 7.88)) with a distance of 0.03	0.003329 seconds
2	1000	The closest points are ((8.96, 3.55), (8.97, 3.55)) with a distance of 0.01	0.171189 seconds
3	10.000	The closest points are ((7.33, 3.72), (7.33, 3.72)) with a distance of 0.00	16.087196 seconds
4	Custom x and y coordonates by user (15)	The closest points are ((5.0, 4.0), (5.0, 4.0)) with a distance of 0.00	0.000205 seconds
5	Random coordonates generated by program function	The closest points are ((1.8, 8.4), (0.95, 9.2)) with a distance of 1.17	0.000070 seconds

Solution 3 – Ridah

ID	Input (Number of coordonates)	Output (Closest Points)	Time taken

1	10.000 points	The shortest distance is 1.0 between points (12.0, -39.0) and (13.0, -39.0)	Time elapsed: 1.1563677787780762 seconds

Appendix C - Evidence of team contribution

All the members of the group have been actively involved in the cw and coding. We all came up with a set of questions in 2-d plane and we chose a question that we were all comfortable with coding then decided to split tasks in between eachother with the report, submitting, team meetings etc.

In case we were not able to meet up in person, we have created a whatsapp group chat where we all talked about how we can all split the tasks between us regarding coding and we used the group chat to post the code to eachother so we can compare the test results.

We all decided on the question and then decided to create the solutions within a set time frame after which we shared all the code with to get a clear insight of the differences between out approaches.