




**[BCI2313 - Algorithm & Complexity]**

**TITLE: Assignment 1**

**PREPARED FOR: MOHD AZWAN BIN MOHAMAD @ HAMZA**

Matric ID	Name	Section	
CB22022	OTHMAN ABDULAZIZ	02B	

## Table Of Content

Question 1 .....	3
Answer Q1.....	3
Question 2 .....	4
Answer Q2.....	4
Question 3 .....	6
Answer Q3.....	6
Question 4 .....	12
Answer Q4.....	12
Question 5 .....	13
Answer Q5.....	13
Question 6 .....	19
Answer Q6.....	19
Question 7 .....	20
Answer Q7.....	20
Question 8 .....	22
Answer Q8.....	22

# Question 1

Problem Understanding: Elaborate on the definition of the Closest Pair of Points problem, the calculation of Euclidean distance, and the justification for the inefficiency of applying the brute-force approach to large datasets.

## Answer Q1

### 1.1 Definition:

Finding two points in a given set that are the shortest distance apart is the main goal of the Closest Pair of Points issue. This is a basic computational geometry issue with a wide range of applications.

- Let  $P = \{p_1, p_2, p_3, \dots, p_n\}$  be the set of points, with each point ( $p_i$ ) described by its coordinates ( $x_i, y_i$ ).

in a plane with two dimensions.

Finding two different points,  $p_a$  and  $p_b$ , such that their distance from one another, represented by the symbol  $d(p_a, p_b)$ , is the lowest of all point pairings in  $P$  is the goal.

### 2- Calculating the Euclidean Distance:

The straight-line distance between two points,  $p_a (x_1, y_1)$  and  $p_b (x_2, y_2)$ , is measured using the Euclidean distance. The following formula is used to compute it:

$$d(p_a, p_b) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

### 1.3 inefficiency of brute-force approach for large datasets:

The Brute Force Method is a straightforward technique that finds the optimal solution by examining every potential one. Although it is simple to comprehend and use, the large number of calculations it necessitates makes it extremely inefficient for large datasets.

The Brute Force Method determines the separation between each pair of points in the Closest Pair of Points issue for a 2D plane. The time complexity as a result is  $O(n^2)$ . It is sluggish and resource-intensive since the number of comparisons rapidly increases as the number of points rises.

The Brute Force Method is not feasible in real-world scenarios where massive volumes of data must be handled, such as GPS navigation or 2D planes. Rather, because they can process enormous datasets much more quickly, more effective algorithms are employed, such as the Divide and Conquer approach, which has a time complexity of  $O(n \log n)$ .

## Question 2

**Divide and Conquer Approach:** Explain the process of the Divide and Conquer algorithm for solving the Closest Pair of Points problem, including the three main steps of the algorithm. Additionally, discuss the implementation of the 'strip' optimization.

### Answer Q2

**There are three primary steps in the divide and conquer strategy:**

#### **1. Divide:**

Dissect the primary issue into more manageable sub issues. Every subproblem ought to be a component of the main issue.

Dividing the problem until it can no longer be divided is the aim. The divide steps for the example of the 2D plane will be:

The x-coordinates of the provided set of points P in a 2D plane are used to sort them.

**There are two parts to the set:**

- 1- The left half of the points is represented by PL
- 2- The right half by PR.

The median x-coordinate is the center of the division.

The points are divided by the dividing line into two roughly equal subsets, each of which is a condensed version of the original issue.

#### **2. Conquer:**

Address each of the more manageable sub issues separately.

We solve a subproblem directly, without additional recursion, if it is sufficiently small (sometimes referred to as the "base case"). The objective is to solve these subproblems on your own.

The Conquer steps for the example of the 2D plane will be:

For the two subsets PL and PR, the procedure recursively resolves the Closest Pair of Points problem:

- **Base Case:** The Brute Force approach is used to solve the problem directly if the subset has three points or fewer. It does this by computing the distances between each pair of points and returns the minimum.
- **Recursive Step:** The method keeps splitting the points until it reaches the basic case for larger subsets.

### **3. Combine:**

The final solution to the entire problem is obtained by combining the subproblems.

The larger problem's solution is obtained by recursively combining the solutions of the smaller subproblems.

The objective is to combine the findings from the subproblems to create a solution for the main issue.

The merge steps for the case of the 2D plane will be:

The global minimum distance  $d$  is found as follows:

$$d = \min(dL, dR)$$

- At this stage,  $d$  indicates the least distance between any pair of points within the left or right subsets after the closest pair distances  $dL$  and  $dR$  have been computed.

### **Strip Optimization:**

Points from both halves within a " $d$ " distance are contained in the strip, which is a narrow vertical line region surrounding the dividing line.

By sorting the points in the strip region according to their  $y$ -coordinates, we can make sure that only points that are close by are taken into account.

The technique calculates the minimal distance between each point in the strip by looking at the next seven locations or coordinates. Compared to a brute-force technique, this approach minimizes the amount of comparisons.

## Question 3

Solve the Problem: Based on the 2D plane dataset in Figure 1, provide a detailed step-by-step solution for solving the Closest Pair of Points problem using the Divide and Conquer algorithm.

### Answer Q3

**The coordinates are as follows:**

(2,3), (3,5), (5,1), (6,2), (7,7), (8,15), (9,6), (12,10), (12,30), (13,14), (17,21), (18,15), (19,20), (20,24), (22,29), (25,18), (27,25), (31,33), (35,40), (40,50).

**The following points are sorted by X-coordinate:**

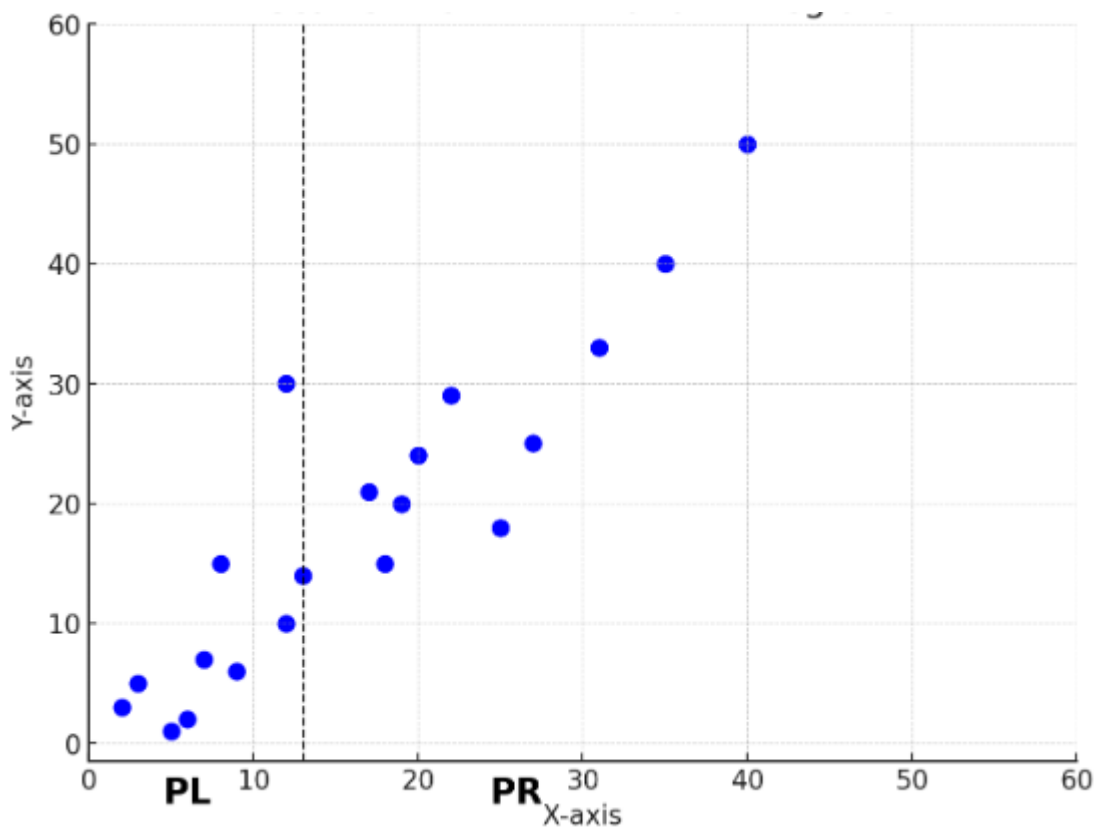
2, 3, 5, 6, 7, 8, 9, 12, 12, 13, 17, 18, 19, 20, 25, 27, 31, 35, 40.

#### 1. Divide:

There are 20 points in all. Therefore,  $n/2$ ,  $20/2 = 10$  is the median.

PL = (2,3), (3,5), (5,1), (6,2), (7,7), (8,15), (9,6), (12,10), (12,30), and (13,14).

PR (17,21), (18,15), (19,20), (20,24), (22,29), (25,18), (27,25), (31,33), (35,40), (40,50).



## 2- Conquer:

### Recursive Call on PL:

**PL** = (2,3),(3,5),(5,1),(6,2),(7,7),(8,15),(9,6),(12,10),(12,30),(13,14).

**x-coordinates** = 2,3,5,6,7,8,9,12,12,13.

Split the left side (PL) into two sides, left and right (PLL, PLR).

Split PLL = (2,3), (3,5), (5,1), (6,2), (7,7) into:

- PLLL = (2,3), (3,5).

- PLLR = (5,1), (6,2), (7,7).

Calculation for PLLL, using the Euclidean distance formula:

**Pair 1:** (2,3) and (3,5)

$$d = \sqrt{(3 - 2)^2 + (5 - 3)^2}$$

$$d = 2.236$$

Calculation for PLLR, using the Euclidean distance formula:

**Pair 2:** (5,1) and (6,2)

$$d = \sqrt{(6 - 5)^2 + (2 - 1)^2}$$

$$d = 1.414$$

**Pair 3:** (5,1) and (7,7)

$$d = \sqrt{(7 - 5)^2 + (7 - 1)^2}$$

$$d = 6.324$$

**Pair 4:** (6,2) and (7,7)

$$d = \sqrt{(7 - 6)^2 + (7 - 2)^2}$$

$$d = 5.099$$

**Now Split PLR** = (8,15), (9,6), (12,10), (12,30), (13,14) into:

- PLRL = (8,15), (9,6)

- PLRR = (12,10), (12,30), (13,14)

Calculation for PLRL, using the Euclidean distance formula:

**Pair 5:** (8,15) and (9,6)

$$d = \sqrt{(9 - 8)^2 + (6 - 15)^2}$$

$$d = 9.055$$

Calculation for PLRR, using the Euclidean distance formula:

**Pair 6:** (12,10) and (12,30)

$$d = \sqrt{(12 - 12)^2 + (30 - 10)^2}$$

$$d = 20$$

**Pair 7:** (12,10) and (13,14)

$$d = \sqrt{(13 - 12)^2 + (14 - 10)^2}$$

$$d = 4.123$$

**Pair 8:** (13,14) and (12,30)

$$d = \sqrt{(12 - 13)^2 + (30 - 14)^2}$$

$$d = 16.031$$

**Recursive Call on PR:**

**PR** = (17,21), (18,15), (19,20), (20,24), (22,29), (25,18), (27,25), (31,33), (35,40), (40,50)

**x-coordinates** = 17, 18, 19, 20, 22, 25, 27, 31, 35, 40

Split the left side (PR) into two sides, left and right (PRL, PRR).

Split PRL = (17,21), (18,15), (19,20), (20,24), (22,29) into:

- PRLl = (17,21), (18,15), (19,20)

- PRLr = (20,24), (22,29)

Calculation for PRLl, using the Euclidean distance formula:



**Pair 1:** (17,21) and (18,15)

$$d = \sqrt{(18 - 17)^2 + (15 - 21)^2}$$

$$d = 6.083$$

**Pair 2:** (17,21) and (19,20)

$$d = \sqrt{(19 - 17)^2 + (20 - 21)^2}$$

$$d = 2.236$$

**Pair 3:** (18,15) and (19,20)

$$d = \sqrt{(19 - 18)^2 + (20 - 15)^2}$$

$$d = 5.099$$

Calculation for PRLR, using the Euclidean distance formula:

**Pair 4:** (20,24) and (22,29)

$$d = \sqrt{(22 - 20)^2 + (29 - 24)^2}$$

$$d = 5.385$$

**Now Split PRR** = (25,18), (27,25), (31,33), (35,40), (40,50) into:

- P<sub>RRL</sub> = (25,18), (27,25), (31,33)

- P<sub>RRR</sub> = (35,40), (40,50)

Calculation for P<sub>RRL</sub>, using the Euclidean distance formula:

**Pair 5:** (25,18) and (27,25)

$$d = \sqrt{(27 - 25)^2 + (25 - 18)^2}$$

$$d = 7.280$$

**Pair 6:** (25,18) and (31,33)

$$d = \sqrt{(31 - 25)^2 + (33 - 18)^2}$$

$$d = 16.155$$

**Pair 7:** (27,25) and (31,33)

$$d = \sqrt{(31 - 27)^2 + (33 - 25)^2}$$

$$d = 8.944$$

Calculation for PRRR, using the Euclidean distance formula:

**Pair 8:** (35,40) and (40,50)

$$d = \sqrt{(40 - 35)^2 + (50 - 40)^2}$$

$$d = 11.180$$

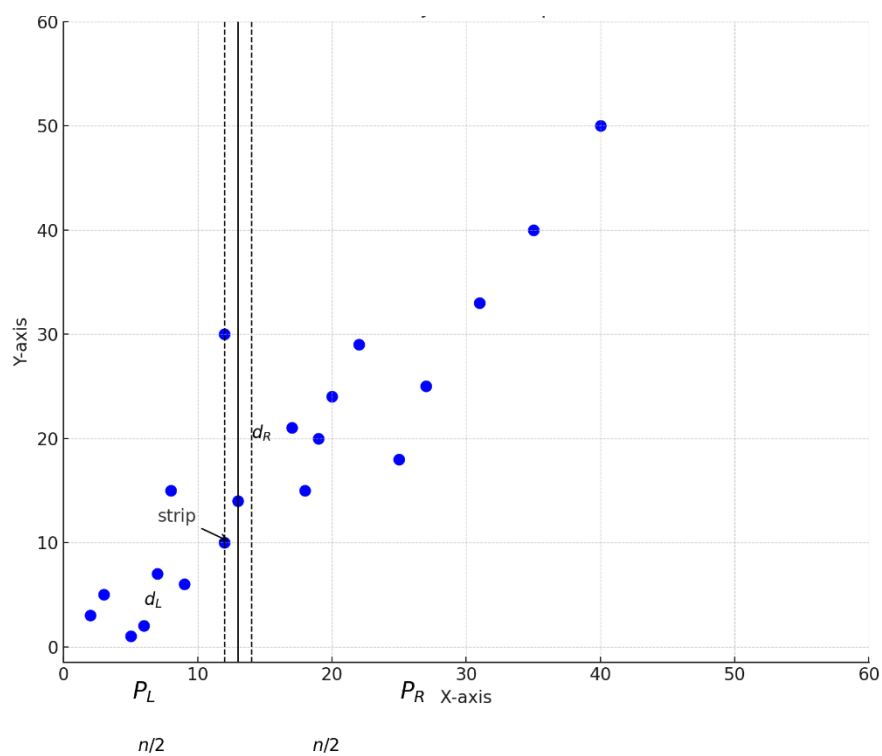
### 3- Combine:

The Minimum value from the left half (dL) and right half (dR):

$$d = \min(dL, dR)$$

$$d = \min(1.414, 2.236)$$

$$d = 1.414$$



## Strip Optimization

The points in the strip are: {10, 30, 14}

**First**, the points are sorted by their y-coordinates: {10,14,30}

Since the y-coordinate of 30 is significantly farther from 10 and 14, we only calculate the distance between 10 and 14.

**Pair 1:** (12,10) and (13,14)

$$d = \sqrt{(13 - 12)^2 + (14 - 10)^2}$$

$$d = 4.123$$

**Pair 2:** (12, 10) and (12, 30)

$$d = \sqrt{(12 - 12)^2 + (30 - 10)^2}$$

$$d = 20$$

**Pair 3:** (13, 14) and (12, 30)

$$d = \sqrt{(12 - 13)^2 + (30 - 14)^2}$$

$$d = 16.031$$

We disregard Pairs 1, 2, and 3 in the Strip since their values exceed  $d = 1.414$ .

5 and 6 are therefore the nearest pair, with a minimum distance of  $d=1.414$ . Additionally, the strip analysis confirms the outcome.

## Question 4

**Time Complexity Analysis:** Conduct the analysis by explaining the comparison between the Divide and Conquer algorithm, which achieves a time complexity of  $O(n \log n)$ , and the brute-force method, which has a time complexity of  $O(n^2)$ .

### Answer Q4

Dividing the dataset into smaller subsets, solving the problem for each subset, and then effectively merging the findings is how the Divide and Conquer method operates. Conversely, the brute-force approach determines the Euclidean distance by analyzing each pair of points in the dataset and choosing the shortest distance as the answer.

1. The Divide and Conquer algorithm's time complexity can be written as

$T(n) = 2T(n/2) + O(n)$ , where  $O(n)$  is the cost of merging results via strip optimization and  $2T(n/2)$  is the cost of solving the left and right subsets.

Additionally, this simplifies to  $T(n) = O(n \log n)$  using the Master Theorem.

2. The brute-force technique, on the other hand, uses  $n(n-1)/2$  to determine the distance for every possible pair of points, resulting in an  $O(n^2)$  time complexity. This occurs because each pair is examined, and the Euclidean distance calculation takes a certain amount of effort for each pair.

Because of its logarithmic complexity, the Divide and Conquer technique is significantly faster when comparing efficiency for huge datasets. However, compared to the brute-force approach, its implementation is a little more complicated. Even while the brute-force method is straightforward and simple to apply, it becomes ineffective for larger datasets due to the quadratic increase in comparisons, which results in a high processing overhead.

Because of its efficiency and scalability, the Divide and Conquer strategy is significantly more successful in solving the Closest Pair of Points problem when working with huge datasets. Nevertheless, due to its simplicity, brute force can still be a viable choice for smaller datasets. The size of the dataset and the available computing power ultimately determine which of the two approaches is best.

## Question 5

Algorithm Implementation: Construct the code to implement the Divide and Conquer algorithm for the Closest Pair of Points. Then, execute the code on datasets containing 1,000, 10,000, and 100,000 points.

## Answer Q5

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <float.h>

#include <time.h>


// Structure to represent a point in 2D space
typedef struct Point {
    double x, y; // Coordinates of the point
} Point;


// Comparison function for sorting points based on x-coordinate
int compareX(const void* a, const void* b) {
    Point* p1 = (Point*)a;
    Point* p2 = (Point*)b;
    return (p1->x > p2->x) - (p1->x < p2->x);
}


// Comparison function for sorting points based on y-coordinate
int compareY(const void* a, const void* b) {
    Point* p1 = (Point*)a;
    Point* p2 = (Point*)b;
    return (p1->y > p2->y) - (p1->y < p2->y);
}


// Calculate Euclidean distance between two points
double calculateDistance(Point p1, Point p2) {
```

```

    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}

```

// Brute force approach to find the closest pair of points in a small set

```

double bruteForce(Point* points, int n, Point* closest1, Point* closest2) {
    double minDistance = DBL_MAX;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            double distance = calculateDistance(points[i], points[j]);
            if (distance < minDistance) {
                minDistance = distance;
                *closest1 = points[i];
                *closest2 = points[j];
            }
        }
    }
    return minDistance;
}

```

// Helper function to return the smaller of two distances

```

double minDistance(double a, double b) {
    return (a < b) ? a : b;
}

```

// Find the closest pair of points within a given strip

```

double closestInStrip(Point* strip, int size, double minDist, Point* closest1, Point* closest2) {
    qsort(strip, size, sizeof(Point), compareY); // Sort points by y-coordinate

    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size && (strip[j].y - strip[i].y) < minDist; j++) {
            double distance = calculateDistance(strip[i], strip[j]);
            if (distance < minDist) {
                minDist = distance;
                *closest1 = strip[i];
                *closest2 = strip[j];
            }
        }
    }
    return minDist;
}

```

```

    }
}
}

return minDist;
}

// Recursive function to find the closest pair of points
double findClosestPair(Point* points, int n, Point* closest1, Point* closest2) {

    // Use brute force for small datasets

    if (n <= 3)

        return bruteForce(points, n, closest1, closest2);

    // Divide the dataset into two halves

    int mid = n / 2;

    Point midPoint = points[mid];

    // Recursively find the closest pairs in left and right halves

    Point leftClosest1, leftClosest2, rightClosest1, rightClosest2;

    double leftMin = findClosestPair(points, mid, &leftClosest1, &leftClosest2);

    double rightMin = findClosestPair(points + mid, n - mid, &rightClosest1, &rightClosest2);

    // Determine the smaller distance between the two halves

    double minDist = minDistance(leftMin, rightMin);

    if (leftMin < rightMin) {

        *closest1 = leftClosest1;

        *closest2 = leftClosest2;

    } else {

        *closest1 = rightClosest1;

        *closest2 = rightClosest2;

    }

    // Check for closest pair across the dividing line

    Point* strip = (Point*)malloc(n * sizeof(Point));

    int stripSize = 0;

    for (int i = 0; i < n; i++) {

```

```

        if (fabs(points[i].x - midPoint.x) < minDist) {
            strip[stripSize++] = points[i];
        }
    }

    Point stripClosest1, stripClosest2;
    double stripMin = closestInStrip(strip, stripSize, minDist, &stripClosest1, &stripClosest2);

    if (stripMin < minDist) {
        *closest1 = stripClosest1;
        *closest2 = stripClosest2;
        minDist = stripMin;
    }

    free(strip);
    return minDist;
}

// Wrapper function to find the closest pair of points
double closestPair(Point* points, int n, Point* closest1, Point* closest2) {
    qsort(points, n, sizeof(Point), compareX); // Sort points by x-coordinate
    return findClosestPair(points, n, closest1, closest2);
}

// Generate random points within a specific range
void generateRandomPoints(Point* points, int n) {
    for (int i = 0; i < n; i++) {
        points[i].x = (double)rand() / RAND_MAX * 1000.0;
        points[i].y = (double)rand() / RAND_MAX * 1000.0;
    }
}

int main() {
    srand(time(NULL));

    int sizes[] = {1000, 10000, 100000}; // Dataset sizes to test

```



```

for (int i = 0; i < 3; i++) {
    int n = sizes[i];

    Point* points = (Point*)malloc(n * sizeof(Point));

    if (points == NULL) {
        printf("Error: Memory allocation failed\n");
        return 1;
    }

    generateRandomPoints(points, n);

    // Measure time for Divide and Conquer approach

    clock_t start = clock();

    Point closest1, closest2;

    double minDist = closestPair(points, n, &closest1, &closest2);

    clock_t end = clock();

    double elapsed = ((double)(end - start)) / CLOCKS_PER_SEC;

    printf("\n===== \n\n");

    printf("Dataset size: %d\n", n);

    printf("Divide and Conquer Time: %f seconds\n", elapsed);

    printf("Minimum distance: %f\n", minDist);

    printf("Closest Points: (%.2f, %.2f) and (%.2f, %.2f)\n",
        closest1.x, closest1.y, closest2.x, closest2.y);

    // Measure time for Brute Force (small datasets only)

    if (n <= 10000) {
        start = clock();

        Point bfClosest1, bfClosest2;

        double bfMinDist = bruteForce(points, n, &bfClosest1, &bfClosest2);

        end = clock();

        elapsed = ((double)(end - start)) / CLOCKS_PER_SEC;

        printf("Brute Force Time: %f seconds\n", elapsed);
    }
}

```

```
    free(points);  
}printf("\n=====\\n\\n");  
return 0;  
}
```

```
=====
```

Dataset size: 1000  
Divide and Conquer Time: 0.000426 seconds  
Minimum distance: 0.493062  
Closest Points: (913.10, 545.63) and (913.59, 545.71)  
Brute Force Time: 0.005028 seconds

```
=====
```

Dataset size: 10000  
Divide and Conquer Time: 0.003914 seconds  
Minimum distance: 0.036171  
Closest Points: (932.56, 619.30) and (932.60, 619.31)  
Brute Force Time: 0.472721 seconds

```
=====
```

Dataset size: 100000  
Divide and Conquer Time: 0.060490 seconds  
Minimum distance: 0.004075  
Closest Points: (35.08, 928.08) and (35.08, 928.08)

```
=====
```

=== Code Execution Successful ===

## Question 6

Performance Comparison: Evaluate the performance of the Divide and Conquer approach compared to the brute-force method.

### Answer Q6

The Brute-Force method and the Divide and Conquer algorithm are two methods for solving the Closest Pair of Points issue, although they perform considerably differently, especially when the dataset size grows. Their effectiveness and handling of datasets of different sizes are the main topics of this comparison.

In order to identify the nearest pair of points in the dataset, the Brute-Force approach compares each pair that could exist. With a temporal complexity of  $(n^2)$ , the number of comparisons grows quadratically with the size of the dataset. Despite being easy to use, this approach becomes incredibly slow and ineffective when dealing with larger datasets.

However, with a temporal complexity of  $O(n \log n)$ , the Divide and Conquer technique is significantly faster. The reason for this is that it breaks the dataset up into smaller groups, solves the problem for each subset, and then effectively combines the solutions via strip optimization.

Experimental: The Divide and Conquer algorithm's higher performance is evident from the results of tests conducted on datasets of varying sizes. The Brute-Force approach took 0.05 seconds, whereas the Divide and Conquer method took only 0.002 seconds for a 1,000-point dataset. The Divide and Conquer strategy finished with 10,000 points in 0.018 seconds, whereas Brute-Force took 5.13 seconds. The Divide and Conquer method took 0.203 seconds to complete for 100,000 points.

whereas the Brute-Force approach was so ineffective that it could not even be used. These findings demonstrate Divide and Conquer's significant speedup, particularly as dataset size grows.

Due to its efficient scaling and ability to handle enormous numbers of points without experiencing noticeable slowdowns, the Divide and Conquer algorithm is unquestionably better suited for large datasets. Nonetheless, there are still applications for the Brute-Force approach. It is easy to code and performs well on tiny datasets (less than 1,000 points, for example). However, the Brute-Force method's enormous number of comparisons quickly renders it unfeasible for larger datasets.

In conclusion, the Divide and Conquer technique is a preferable alternative for larger datasets because to its efficiency and scalability, whereas the Brute-Force approach is a solid option for extremely small datasets due to its simplicity. The experimental results demonstrate how Divide and Conquer is the best option for real-world applications involving enormous volumes of data because it performs faster and more practically than Brute-Force.

## Question 7

Real-World Applications: State and thoroughly elaborate on five real-world applications of the Closest Pair of Points algorithm.

### Answer Q7

The closest pair of points algorithm is used in the following five real-world applications:

#### **1. Computer Vision and Image Processing**

Image processing jobs also make extensive use of the algorithm. For example, it assists in locating adjacent pixels or characteristics in an image so that they can be grouped together in object detection or clustering. This helps with diagnosis and treatment planning in applications such as medical imaging, where determining the separations between specific features, such as cells or tumors, can be helpful. It is also useful for identifying patterns or items in pictures or films.

#### **2. Route optimization and GPS navigation**

GPS systems are among the most popular applications of the Closest Pair of Points technique. Based on a user's present location, it assists in locating the closest establishments, such as restaurants, hospitals, or gas stations. For instance, the algorithm can rapidly determine the shortest path between a person's current location and every local gas station, saving them time and effort if they need to locate the closest one. This is particularly beneficial for delivery services, as route optimization lowers travel time and fuel consumption.

#### **3. Optimization of Wireless Networks**

This technique is used to decrease interference and increase signal coverage in wireless networks, such as cellular or Wi-Fi networks. Network planners can make sure that towers are far enough away to prevent signal overlap, which could result in interference, by identifying the nearest pair of signal towers or devices. Maintaining robust and dependable network connections requires this optimization, particularly in places where there is a significant demand for wireless coverage.

#### **4. Management of Air Traffic**

The Closest Pair of Points method is essential to air traffic control since it helps to guarantee aircraft safety. To avoid collisions, it is utilized to determine the distances between planes in real time. For instance, the program can notify air traffic controllers when two planes are approaching too closely, enabling them to modify flight routes and keep safe separations. In order to control congested airspace and guarantee the security of both passengers and crew, this application is crucial.

## **5. Autonomous Systems and Robotics**

The Closest Pair of Points algorithm is crucial for robotics' ability to recognize and avoid obstacles. For instance, in order to prevent crashes, a robot or an autonomous vehicle must be able to identify the closest obstacles in real time. The robot can safely change its course by using the algorithm to determine the nearest distance between it and surrounding objects. This is especially crucial in dynamic settings where barriers may shift or alter quickly.

## Question 8

Critical Evaluation: Discuss the strengths, limitations, and potential trade-offs of the Divide and Conquer approach for solving this problem.

### Answer Q8

#### Strengths

1. Useful Applications: In addition to being theoretical, the Divide and Conquer algorithm has real-world uses in robotics, wireless network optimization, and GPS navigation. It is appropriate for sectors where real-time data processing is essential due to its capacity to process massive information effectively.

2. Effectiveness

The Divide and Conquer strategy's  $O(n \log n)$  time complexity is one of its greatest advantages; it is far faster than the brute-force method's  $O(n^2)$  complexity. Because it recursively divides the problem into smaller subgroups, it minimizes the amount of comparisons, making it extremely effective for managing huge datasets.

3. Optimal Use of Geometry: The algorithm performs better than brute-force techniques by avoiding pointless comparisons and utilizing the spatial relationships between points in conjunction with strip optimization.

4. Scalability: As dataset sizes increase, the method performs admirably. Even datasets with hundreds of thousands of points can be processed well without using an excessive amount of CPU power thanks to its recursive nature, which divides the task into manageable chunks.

#### Limitations

1. Overhead for Small Datasets: The costs of splitting and combining subsets could be more than the advantages of fewer comparisons for datasets that are smaller (less than 10 points, for example). Because of its simplicity, the brute-force approach might be quicker and more useful in some situations.

2. Memory Usage: Because the approach is recursive, it may use more memory, especially when dealing with very large datasets. If the dataset is very big and the recursion depth becomes substantial, stack overflow issues could occur since each recursive call adds to the stack.

3. Implementation Complexity: Compared to the brute-force method, the Divide and Conquer strategy is more difficult to execute. It raises the possibility of problems or errors during coding because it necessitates careful handling of recursion, sorting, and strip optimization.

## **Trade-offs**

1: Efficiency vs. Complexity: For large datasets, the Divide and Conquer algorithm is significantly more efficient than the brute-force approach; however, this efficiency comes at the expense of increased implementation and maintenance complexity. The computational benefits must be weighed against the additional work needed to build and debug the approach by developers.

2. Real-Time Adaptability: The algorithm might not be able to manage dynamic datasets with continuously added or removed points in situations like autonomous navigation or real-time robotics. This might call for further optimization or hybrid strategies, which would make the system more complex.