Github repository for python implementation of the divide and conquer algorithm and the graphical representation of the problem using the matplotlib library: https://github.com/CosteanRobert/closestPairProblem

The closest pair problem is a classical problem in computer science that involves finding the two points in a set of points with the smallest distance between them. This problem has numerous applications in areas such as image processing, computer graphics, and geographic information systems. In this documentation, we will explore the different approaches to solving the closest pair problem, including the brute force method and the divide and conquer method. We will discuss the time complexity of these approaches and compare their performance. We will also discuss the practical applications of the closest pair problem and how it can be used in real-world scenarios.

The brute force method for solving the closest pair problem involves comparing the distance between every pair of points in the set and finding the minimum distance. This method has a time complexity of O(n^2), where n is the number of points in the set. To implement the brute force method, we can define a function that takes a list of points as input and returns the minimum distance and pair of points. The function will iterate through the list of points and calculate the distance between each pair of points using the Euclidean distance formula: d = sqrt[(x1 - x2)^2 + (y1 – y2)^2] where (x1, y1) and (x2, y2) are the coordinates of the two points. The function can then compare the distance between each pair of points and keep track of the minimum distance and corresponding pair of points. Once the function has compared all pairs of points, it can return the minimum distance and pair of points as the result. While the brute force method is simple to implement, it has a relatively high time complexity and may not be suitable for large sets of points.

The divide and conquer method is a more efficient approach to solving the closest pair problem, with a time complexity of O(n log n). This method works by dividing the set of points into two halves and recursively solving the problem on each half. The subproblems are then combined to find the overall solution. To find the minimum distance in each subproblem, the divide and conquer approach uses a three-step process: Find the median of the x-coordinates of the points. Divide the points into two halves based on their x-coordinates, with one half on either side of the median. Find the minimum distance between the points in each half, and also the minimum distance between points that straddle the median. We consider that a point is straddling the median if the points is at most δ distance away from the middle line dividing the two sets where δ is the minimum of the 2 distances calculated for each half. The minimum distance between points that straddle the median can be found in O(n) time by sorting the points by their y-coordinates and comparing the distances between the points in a sliding window.

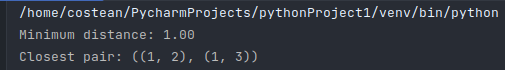
Once the minimum distances for each subproblem have been found, the overall minimum distance can be obtained by comparing the minimum distances of the subproblems and taking the minimum of the three values. To calculate the time complexity of the divide and conquer approach, we consider the number of comparisons that are made at each step. In the first step, the points are divided into two halves, which requires O(n) comparisons. In the second step, the minimum distance is found in each half and between points that straddle the median, which requires O(n log n) comparisons. Therefore, the overall time complexity of the divide and conquer approach is O(n + n log n) = O(n log n).

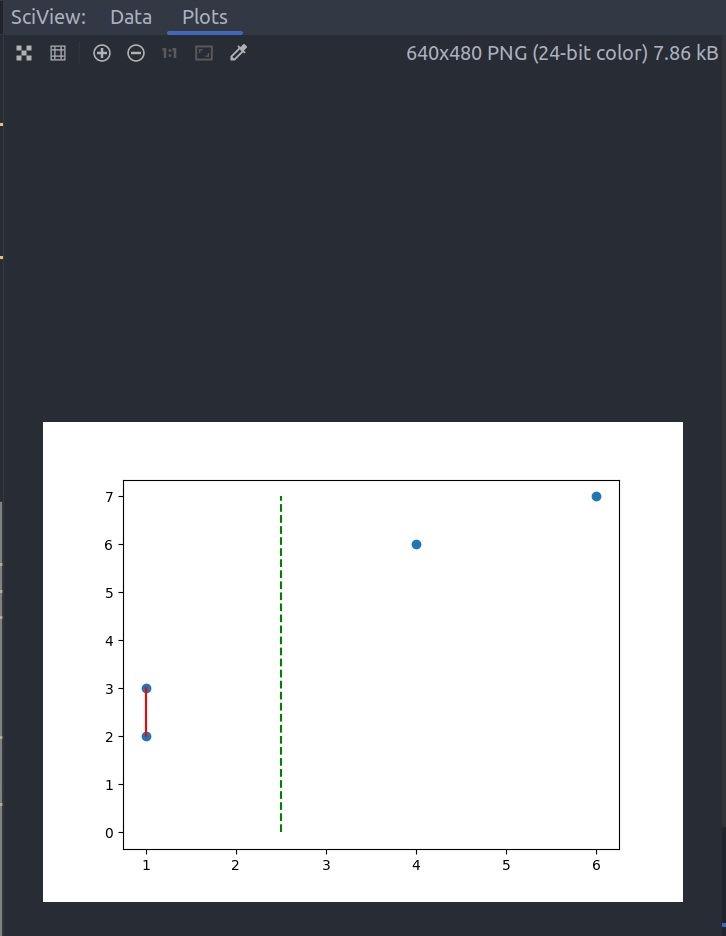
The time complexity of the divide and conquer approach is O(n log n), which is significantly faster than the O(n^2) time complexity of the brute force method for large sets of points. However, it requires more complex code to implement, and it may not be the most efficient solution for small sets of points.

The closest pair problem has numerous practical applications in various fields. Here are a few examples: Image processing: In image processing, the closest pair problem can be used to identify objects or features in an image by finding the closest pair of points that belong to the same object or feature. Computer graphics: In computer graphics, the closest pair problem can be used to find the shortest path between two points on a map or in a virtual environment. It can also be used to find the closest pair of points in a set of 3D points, which can be useful for various purposes such as rendering or collision detection. Geographic information systems: In geographic information systems, the closest pair problem can be used to find the nearest neighbor of a point, such as the nearest hospital or gas station. It can also be used to find the shortest route between two points on a map, such as the fastest way to get from one location to another. Finance: In finance, the closest pair problem can be used to identify pairs of stocks that are highly correlated, or to find the nearest ATM or bank branch.

In conclusion, the closest pair problem is a classical problem in computer science that involves finding the two points in a set of points with the smallest distance between them. The brute force method has a time complexity of O(n^2) and is simple to implement, but may not be suitable for large sets of points. The divide and conquer method has a time complexity of O(n log n) and is more efficient, but requires more complex code to implement. The closest pair problem has numerous practical applications in areas such as image processing, computer graphics, and geographic information systems.

Python program result for the [(1, 2), (1, 3), (4, 6), (6, 7)] point set.

Matplotlib grahpical representation of the same set:



Another set: [(4, 6), (6, 7), (3, 8), (5, 9)]

