Minimum Distance between Two Points - GeeksforGeeks

Source: https://www.geeksforgeeks.org/closest-pair-of-points/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Minimum Distance between Two Points Last Updated : 23 Jul, 2025 You are given an array arr[] of n distinct points in a 2D plane, where each point is represented by its (x, y) coordinates. Find the minimum Euclidean distance between two distinct points . Note: For two points A(p x ,q x) and B(p y ,q y) the distance Euclidean between them is: Distance = \sqrt{(p_{x}-q_{x})^{2}+ (p_{y}-q_{y})^{2}} Examples: Input : arr[] = [[-1, -2], [0, 0], [1, 2], [2, 3]] Output : 1.414214 Explanation : The smallest distance is between points (1, 2) and (2, 3), which is 1.414214. Input : arr[] = [[-2, -2], [1, 2], [-1, 0], [3, 3]] Output: 2.236068 Explanation : The smallest distance is between points (-2, -2) and (-1, 0), which is 2.236068. Table of Content [Naive Approach] Checking All Pairs - O(n^2) Time and O(1) Space [Expected Approach] Using Divide and Conquer - O(n log(n)) Time and O(n) Space [Naive Approach] Checking All Pairs - O(n^2) Time and O(1) Space The idea is to check the distance between all pairs of points and store the minimum distance found. C++ // C++ program to find closet point #include <iostream> #include <vector> #include <cmath> #include <iomanip> using namespace std ; // Function to compute Euclidean distance between two points double distance (const vector < double >& p1 , const vector < double >& p2) { return sqrt ((p1 [0] - p2 [0]) * (p1 [0] - p2 [0]) + (p1 [1] - p2 [1]) * (p1 [1] - p2 [1])); } // Function that returns the smallest distance // between any pair of points double minDistance (const vector < vector < double >>& points) { int n = points . size (); double minDist = 1e9 ; // Brute force to check all pairs for (int i = 0 ; i < n ; ++ i) { for (int j = i + 1 ; j < n ; ++ j) { double dist = distance (points [i], points [j]); if (dist < minDist) { minDist = dist ; } } } // Return the smallest distance return minDist ; } int main () { vector < vector < double >> points = {{ -1 , -2 }, { 0 , 0 }, { 1 , 2 }, { 2 , 3 }}; double res = minDistance (points); cout << fixed << setprecision (6) << res << endl ; return 0 ; } Java // Java program to find closest point import java.util.ArrayList ; import java.util.List ; import java.lang.Math ; class GfG { // Function to compute Euclidean distance between two points static double distance (double [] p1 , double [] p2) { return Math . sqrt ((p1 [0] - p2 [0]) * (p1 [0] - p2 [0]) + (p1 [1] - p2 [1]) * (p1 [1] - p2 [1])); } // Function that returns the smallest distance // between any pair of points static double minDistance (List < double []> points) { int n = points . size (); double minDist = Double . MAX_VALUE ; // Brute force to check all pairs for (int i = 0 ; i < n ; ++ i) { for (int j = i + 1 ; j < n ; ++ j) { double dist = distance (points . get (i), points . get (j)); if (dist < minDist) { minDist = dist ; } } } // Return the smallest distance return minDist ; } public static void main (String [] args) { List < double []> points = new ArrayList <> (); points . add (new double [] { - 1 , - 2 }); points . add (new double [] { 0 , 0 }); points . add (new double [] { 1 , 2 }); points . add (new double [] { 2 , 3 }); double res = minDistance (points); System . out . printf ("%.6f\n" , res); } } Python # Python program to find closet point import math # Function to compute Euclidean distance between two points def distance (p1 , p2): return math . sqrt ((p1 [0] - p2 [0]) ** 2 + (p1 [1] - p2 [1]) ** 2) # Function that returns the smallest distance # between any pair of points def minDistance (points): n = len (points) minDist = float ('inf') # Brute force to check all pairs for i in range (n): for j in range (i + 1 , n): dist = distance (points [i], points [j]) if dist < minDist : minDist = dist # Return the smallest distance return minDist if __name__ == "__main__" : points = [[- 1 , - 2], [0 , 0], [1 , 2], [2 , 3]] res = minDistance (points) print (f " { res : .6f } ") C# // C# program to find closest point using System ; using System.Collections.Generic ; class GfG { // Function to compute Euclidean distance between two points static double distance (double []

p1 , double [] p2) { return Math . Sqrt ((p1 [0] - p2 [0]) * (p1 [0] - p2 [0]) + (p1 [1] - p2 [1]) * (p1 [1] - p2 [1])); } // Function that returns the smallest distance // between any pair of points static double minDistance (List < double [] > points) { int n = points . Count ; double minDist = double . MaxValue ; for (int i = 0 ; i < n ; ++ i) { for (int j = i + 1 ; j < n ; ++ j) { double dist = distance (points [i], points [j]); if (dist < minDist) { minDist = dist ; } } } // Return the smallest distance return minDist ; } static void Main () { List < double [] > points = new List < double [] > { new double [] { - 1 , - 2 }, new double [] { 0 , 0 }, new double [] { 1 , 2 }, new double [] { 2 , 3 } }; double res = minDistance (points); Console . WriteLine (res . ToString ("F6")); } } JavaScript // JavaScript program to find closest point // Function to compute Euclidean distance between two points function distance (p1 , p2) { return Math . sqrt ((p1 [0] - p2 [0]) ** 2 + (p1 [1] - p2 [1]) ** 2); } // Function that returns the smallest distance // between any pair of points function minDistance (points) { let n = points . length ; let minDist = Infinity ; for (let i = 0 ; i < n ; ++ i) { for (let j = i + 1 ; j < n ; ++ j) { let dist = distance (points [i], points [j]); if (dist < minDist) { minDist = dist ; } } } // Return the smallest distance return minDist ; } // Driver Code const points = [[- 1 , - 2], [0 , 0], [1 , 2], [2 , 3]]; const res = minDistance (points); console . log (res . toFixed (6)); Output 1.414214 [Expected Approach] Using Divide and Conquer - O(n log(n)) Time and O(n) Space The main idea is to use the divide and conquer algorithm, where the points are recursively divided into smaller groups. The minimum distance is calculated within each groups , and during the merging step, we will check for possible closer pairs across the dividing point . This approach reduces the number of comparisons , as it only focuses on relevant points near the divide . Step By Step Implementation: Sort the points by x-coordinate . Recursively divide the points into left and right subarrays until each subarrays has one or two points: If one point, return infinity. If two points, return their distance. Find the minimum distances dl and dr in the left and right subarrays , and set d as the smaller value between dl and dr . Build a strip : collect points whose x-distance from the midline is ≤ d . Sort the strip by y-coordinate. For each point in the strip, compare it with the next up to 7 points (having y-distance ≤ d) to check for closer pairs. Update d if a smaller distance is found in the strip. Return the smallest distance found from the left half, right half, or across the strip. Key Insights: Let the dividing point be at coordinate (x, y) . After recursively finding the minimum distance d from the left and right halves, we focus on points near the dividing point that could potentially form a closer pair. We stores all points whose x-distance from the dividing point is ≤ d , i.e., points between x - d and x + d . So, all these points lie inside a vertical strip of width 2d along the x-axis. Why only consider points within x - d to x + d ? If two points lie farther than d apart in the x-direction, their distance is already greater than d , so they can’t be closer . Now, to reduce unnecessary comparisons, we sort these points by their y-coordinate . For each point in this strip, we only compare it with points that are within d units vertically (i.e., y-distance ≤ d). Why compare points with y distance ≤ d ? Because any pair with a y-distance > d will have a total distance > d , so we can safely skip those. This means for any point in the strip[] , we’re only checking points inside a rectangle of dimensions 2d × d (width 2d along x , height d along y). For each point in the strip[], we check at most 7 points, as there are at most 7 points within this rectangle that could potentially have a smaller distance. Why at most 7 comparisons per point in strip[]? Let’s look at this 2d × d rectangle more closely: Split it into two d × d squares : one for points from the left half, one for the right half. Each square is then divided into 4 smaller squares of size (d/2 × d/2) . The diagonal of these smaller squares is less than d/√2 which is less than d , so no two points can occupy the same small square (otherwise, they’d be closer than d , violating the earlier recursive result). So, each d × d square can have at most 4 points , totaling at most 8 points in the full 2d × d rectangle. Since we’re only comparing the current point with the others, that means at most 7 valid comparisons . C++ // C++ program to find minimum distance between points #include <iostream> #include <vector> #include <cmath> #include <algorithm> #include <iomanip> using namespace std ; // Function to compute Euclidean distance between two points double distance (const vector < double >& p1 , const vector < double >& p2) { return sqrt ((p1 [0] - p2 [0]) * (p1 [0] - p2 [0]) + (p1 [1] - p2 [1]) * (p1 [1] - p2 [1])); } // Comparison function to sort points by x-coordinate bool compareX (const vector < double >& p1 , const vector < double >& p2) { return p1 [0] < p2 [0]; } // Comparison function to sort points by y-coordinate bool compareY (const vector < double >& p1 , const vector < double >& p2) { return p1 [1] < p2 [1]; } // Function to find the minimum distance in the strip double stripClosest (vector < vector < double >>& strip , double d) { double minDist = d ; // Sort points in the strip by their y-coordinate sort (strip . begin (), strip . end (), compareY); // Compare each point in the strip for (int i = 0 ; i < strip . size (); ++ i) { // At most 7 times this will run for (int j = i + 1 ; j < strip . size () && (strip [j][1] - strip [i][1]) < minDist ; ++ j) { minDist = min (minDist , distance (strip [i], strip [j])); } } return minDist ; } // Divide and conquer function to find the minimum distance double minDistUtil (vector < vector < double >>&

points , int left , int right) { // Base case brute force for 2 or fewer points if (right - left <= 2) { double minDist = 1e9 ; for (int i = left ; i < right ; ++ i) { for (int j = i + 1 ; j < right ; ++ j) { minDist = min (minDist , distance (points [i], points [j])); } } return minDist ; } // Find the midpoint int mid = (left + right) / 2 ; double midX = points [mid][0]; // Recursively find the minimum distances in // the left and right halves double dl = minDistUtil (points , left , mid); double dr = minDistUtil (points , mid , right); double d = min (dl , dr); // Build the strip of points within distance d from the midline int k = 0 ; vector < vector < double >> strip ; for (int i = left ; i < right ; ++ i) { if (abs (points [i][0] - midX) < d) { strip . push_back (points [i]); } } // Find the minimum distance in the strip double stripDist = stripClosest (strip , d); return min (d , stripDist); } // Function to find the closest pair of points double minDistance (vector < vector < double >>& points) { int n = points . size (); // Sort points by x-coordinate sort (points . begin (), points . end (), compareX); return minDistUtil (points , 0 , n); } int main () { vector < vector < double >> points = {{ -1 , -2 }, { 0 , 0 }, { 1 , 2 }, { 2 , 3 }}; double res = minDistance (points); // Output the result with 6 decimal places cout << fixed << setprecision (6) << res << endl ; return 0 ; } Java // Java program to find minimum distance between points import java.util.* ; import java.lang.Math ; public class GfG { // Function to compute Euclidean distance between two points static double distance (double [] p1 , double [] p2) { return Math . sqrt ((p1 [0] - p2 [0]) * (p1 [0] - p2 [0]) + (p1 [1] - p2 [1]) * (p1 [1] - p2 [1])); } // Comparison function to sort points by x-coordinate static Comparator < double []> compareX = new Comparator < double []> () { public int compare (double [] p1 , double [] p2) { return Double . compare (p1 [0] , p2 [0]); } }; // Comparison function to sort points by y-coordinate static Comparator < double []> compareY = new Comparator < double []> () { public int compare (double [] p1 , double [] p2) { return Double . compare (p1 [1] , p2 [1]); } }; // Function to find the minimum distance in the strip static double stripClosest (double [][] strip , double d) { double minDist = d ; // Sort points in the strip by their y-coordinate Arrays . sort (strip , compareY); // Compare each point in the strip for (int i = 0 ; i < strip . length ; i ++) { for (int j = i + 1 ; j < strip . length && (strip [j][1] - strip [i][1]) < minDist ; j ++) { minDist = Math . min (minDist , distance (strip [i] , strip [j])); } } return minDist ; } // Divide and conquer function to find the minimum distance static double minDistUtil (double [][] points , int left , int right) { // Base case brute force for 2 or fewer points if (right - left <= 2) { double minDist = Double . MAX_VALUE ; for (int i = left ; i < right ; i ++) { for (int j = i + 1 ; j < right ; j ++) { minDist = Math . min (minDist , distance (points [i] , points [j])); } } return minDist ; } // Find the midpoint int mid = (left + right) / 2 ; double midX = points [mid][0] ; // Recursively find the minimum distances in // the left and right halves double dl = minDistUtil (points , left , mid); double dr = minDistUtil (points , mid , right); double d = Math . min (dl , dr); // Build the strip of points within distance d from the midline List < double []> strip = new ArrayList <> (); for (int i = left ; i < right ; i ++) { if (Math . abs (points [i][0] - midX) < d) { strip . add (points [i]); } } // Find the minimum distance in the strip double stripDist = stripClosest (strip . toArray (new double [strip . size ()][]), d); return Math . min (d , stripDist); } // Function to find the closest pair of points static double minDistance (double [][] points) { int n = points . length ; // Sort points by x-coordinate Arrays . sort (points , compareX); return minDistUtil (points , 0 , n); } public static void main (String [] args) { double [][] points = {{ - 1 , - 2 }, { 0 , 0 }, { 1 , 2 }, { 2 , 3 }}; double res = minDistance (points); // Output the result with 6 decimal places System . out . printf ("%.6f\n" , res); } } Python # Python program to find minimum distance between points import math # Function to compute Euclidean distance between two points def distance (p1 , p2): return math . sqrt ((p1 [0] - p2 [0]) ** 2 + (p1 [1] - p2 [1]) ** 2) # Function to find the minimum distance in the strip def stripClosest (strip , d): min_dist = d # Sort points in the strip by their y-coordinate strip . sort (key = lambda point : point [1]) # Compare each point in the strip for i in range (len (strip)): for j in range (i + 1 , len (strip)): if (strip [j][1] - strip [i][1]) < min_dist : min_dist = min (min_dist , distance (strip [i], strip [j])) else : break return min_dist # Divide and conquer function to find the minimum distance def minDistUtil (points , left , right): # Base case brute force for 2 or fewer points if right - left <= 2 : min_dist = float ('inf') for i in range (left , right): for j in range (i + 1 , right): min_dist = min (min_dist , distance (points [i], points [j])) return min_dist # Find the midpoint mid = (left + right) // 2 mid_x = points [mid][0] # Recursively find the minimum distances # in the left and right halves dl = minDistUtil (points , left , mid) dr = minDistUtil (points , mid , right) d = min (dl , dr) # Build the strip of points within distance d from the midl strip = [] for i in range (left , right): if abs (points [i][0] - mid_x) < d : strip . append (points [i]) # Find the minimum distance in the strip stripDist = stripClosest (strip , d) return min (d , stripDist) # Function to find the closest pair of points def minDistance (points): n = len (points) # Sort points by x-coordinate points . sort (key = lambda point : point [0]) return minDistUtil (points , 0 , n) if __name__ == '__main__' : points = [[- 1 , - 2], [0 , 0], [1 , 2], [2 , 3]] res = minDistance (points) # Output the result with 6 decimal places print (f ' { res : .6f }

') C# // C# program to find minimum distance between points using System ; using System.Linq ; using static System . Math ; using System.Collections.Generic ; class GfG { // Function to compute Euclidean distance between two points static double distance (double [] p1 , double [] p2){ double dx = p1 [0] - p2 [0]; double dy = p1 [1] - p2 [1]; return Sqrt (dx * dx + dy * dy); } // Function to find the minimum distance in the strip static double stripClosest (double [][] strip , double d){ double minDist = d ; // Sort points in the strip by their y-coordinate Array . Sort (strip , (p1 , p2) => p1 [1]. CompareTo (p2 [1])); for (int i = 0 ; i < strip . Length ; ++ i){ // The inner loop runs for at most 7 points for (int j = i + 1 ; j < strip . Length && (strip [j][1] - strip [i][1]) < minDist ; ++ j){ minDist = Min (minDist , distance (strip [i], strip [j])); } } return minDist ; } // Divide and conquer function to find the minimum distance static double minDistUtil (double [][] points , int left , int right){ // Base case there are 2 or fewer points if (right - left <= 2){ if (right - left <= 0) return double . MaxValue ; return distance (points [left], points [right - 1]); } // Find the midpoint index int midIndex = left + (right - left) / 2 ; double midX = points [midIndex][0]; // Recursively find the minimum distances // in the left and right halves double dl = minDistUtil (points , left , midIndex); double dr = minDistUtil (points , midIndex , right); double d = Min (dl , dr); // Build the strip of points whose x-coordinate // is within distance 'd' from the mid List < double [] > stripList = new List < double [] > (); for (int i = left ; i < right ; ++ i){ if (Abs (points [i][0] - midX) < d){ stripList . Add (points [i]); } } double [][] stripArray = stripList . ToArray (); // Find the minimum distance in the strip double stripDist = stripClosest (stripArray , d); return Min (d , stripDist); } // Function to find the closest pair of points static double minDistance (double [,] points2D){ int n = points2D . GetLength (0); double [][] points = new double [n][]; for (int i = 0 ; i < n ; i ++){ points [i] = new double [] { points2D [i , 0], points2D [i , 1] }; } // Sort points by x-coordinate Array . Sort (points , (p1 , p2) => p1 [0]. CompareTo (p2 [0])); return minDistUtil (points , 0 , n); } static void Main (){ double [,] points = { { - 1 , - 2 }, { 0 , 0 }, { 1 , 2 }, { 2 , 3 }, { 5 , 1 }, { 6 , 3 }, { 8 , 0 }, { 9 , 2 } }; double res = minDistance (points); // Output the result with 6 decimal places Console . WriteLine (res . ToString ("F6")); } } JavaScript // JavaScript program to find minimum distance between points // Function to compute Euclidean distance between two points function distance (p1 , p2) { return Math . sqrt ((p1 [0] - p2 [0]) ** 2 + (p1 [1] - p2 [1]) ** 2); } // Comparison function to sort points by x-coordinate function compareX (p1 , p2) { return p1 [0] - p2 [0]; } // Comparison function to sort points by y-coordinate function compareY (p1 , p2) { return p1 [1] - p2 [1]; } // Function to find the minimum distance in the strip function stripClosest (strip , d) { let minDist = d ; // Sort points in the strip by their y-coordinate strip . sort (compareY); // Compare each point in the strip for (let i = 0 ; i < strip . length ; i ++) { // At most 7 times this will run for (let j = i + 1 ; j < strip . length && (strip [j][1] - strip [i][1]) < minDist ; j ++) { minDist = Math . min (minDist , distance (strip [i], strip [j])); } } return minDist ; } // Divide and conquer function to find the minimum distance function minDistUtil (points , left , right) { // Base case brute force for 2 or fewer points if (right - left <= 2) { let minDist = Infinity ; for (let i = left ; i < right ; i ++) { for (let j = i + 1 ; j < right ; j ++) { minDist = Math . min (minDist , distance (points [i], points [j])); } } return minDist ; } // Find the midpoint let mid = Math . floor ((left + right) / 2); let midX = points [mid][0]; // Recursively find the minimum distances in the left and right halves let dl = minDistUtil (points , left , mid); let dr = minDistUtil (points , mid , right); let d = Math . min (dl , dr); // Build the strip of points within distance d from the midline let strip = []; for (let i = left ; i < right ; i ++) { if (Math . abs (points [i][0] - midX) < d) { strip . push (points [i]); } } // Find the minimum distance in the strip let stripDist = stripClosest (strip , d); return Math . min (d , stripDist); } // Function to find the closest pair of points function minDistance (points) { let n = points . length ; // Sort points by x-coordinate points . sort (compareX); return minDistUtil (points , 0 , n); } // Driver Code let points = [[- 1 , - 2], [0 , 0], [1 , 2], [2 , 3]]; let res = minDistance (points); // Output the result with 6 decimal places console . log (res . toFixed (6)); Output 1.414214 Applications: Used to detect planes that are too close to each other, preventing potential collisions. Used in motion planning to avoid obstacles by determining the closest obstacles or points in the robot's path. Helps in data classification and pattern recognition by finding nearest data points for training models. Optimizing the placement of devices by finding nearest neighbors for improved communication. Comment Article Tags: Article Tags: Divide and Conquer Geometric DSA Closest Pair of Points