**Closest Pair of Points algorithm**

The Closest Pair of Points algorithm works by employing a divide-and-conquer strategy to find the pair of points with the smallest Euclidean distance among a set of points in a 2D plane.

Here's a step-by-step explanation of how the algorithm works:

1. **Sort Points by X-coordinate**: The algorithm starts by sorting the input points based on their X-coordinates. This step ensures that points are processed in a consistent order and simplifies the subsequent steps.
2. **Divide the Points**: The sorted points are divided into two halves. This is a recursive step where each half is further divided until a base case is reached (usually when there are only a few points left).
3. **Find the Closest Pair in Each Half**: For each half, the algorithm recursively finds the closest pair of points. This is done by applying the same divide-and-conquer strategy to each half.
4. **Merge Step**: After finding the closest pairs in each half, the algorithm merges the two halves and considers the points within a strip of width **2d** centered at the middle point, where **d** is the minimum distance found in the previous steps.
5. **Check the Strip**: The algorithm checks the points within the strip to find any pairs of points that are closer than **d**. This is done by comparing the distances between adjacent points in the strip.
6. **Return the Minimum Distance**: Finally, the algorithm returns the minimum distance **d** found in the previous steps, which represents the closest pair of points in the entire set.

The key idea behind the algorithm is that the closest pair of points either lies entirely in one of the halves or crosses the dividing line between the two halves. By recursively applying the divide-and-conquer strategy and efficiently handling the merging step, the algorithm is able to find the closest pair of points efficiently with a time complexity of O(n log n).

A diagram of a graph

Description automatically generated

A diagram of a diagram

Description automatically generated

Certainly! Let's walk through the example step by step using the given array of points:

Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};

1. **Sorting by X-coordinate**: We first sort the points based on their x-coordinates. This is done to simplify the problem by ensuring that we only need to consider points within a certain range when comparing distances. After sorting, the array becomes:

Point P[] = {{2, 3}, {3, 4}, {5, 1}, {12, 10}, {12, 30}, {40, 50}};

1. **Divide and Conquer**: We divide the array into two halves and recursively find the closest pair of points in each half.

a. **Left Half**: We consider the points **{2, 3}**, **{3, 4}**, **{5, 1}**.

b. **Right Half**: We consider the points **{12, 10}**, **{12, 30}**, **{40, 50}**.

1. **Find the Minimum Distance**: We find the minimum distance between points in the left half (**dl**) and the minimum distance between points in the right half (**dr**).

a. **Left Half**: The minimum distance is between **{2, 3}** and **{3, 4}**, which is approximately **1.414**.

b. **Right Half**: The minimum distance is between **{12, 10}** and **{12, 30}**, which is approximately **20**.

1. **Merge Step**: We merge the two halves back together and consider the points within a strip of width **2d** centered at the middle point. In this case, **d = min(dl, dr) = 1.414**.

a. **Strip**: We consider the points **{2, 3}**, **{3, 4}**, **{5, 1}**, **{12, 10}**, **{12, 30}**.

1. **Check the Strip**: We check for points in the strip that are closer than **d**. In this case, we find that the closest pair of points in the strip is **{2, 3}** and **{3, 4}**, which has a distance of **1.414**.
2. **Result**: The closest pair of points in the entire array is **{2, 3}** and **{3, 4}**, with a distance of **1.414**.

This algorithm has a time complexity of O(n log n) due to the sorting step and the recursive divide-and-conquer steps.

Code:

**struct Point {**

**int x, y;**

**};**

Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};

// Function to find the closest pair of points

double closestPairOfPoints(Point P[], int n) {

if (n <= 3) Apply bruteForce

// Sort the points based on their x-coordinates

sort(P, P + n, compareX);

bool compareX(Point p1, Point p2) {

return p1.x < p2.x;

}

// Find the middle point

int mid = n / 2;

Point midPoint = P[mid];

// Divide the points into two halves

Point left[mid], right[n - mid];

for (int i = 0; i < mid; i++) left[i] = P[i];

for (int i = mid; i < n; i++) right[i - mid] = P[i];

// Recursively find the closest pair of points in each half

double dl = closestPairOfPoints(left, mid);

double dr = closestPairOfPoints(right, n - mid);

// Find the minimum distance between points in the two halves

double d = min(dl, dr);

// Create an array to store the points within the strip of width 2d centered at the middle point

Point strip[n];

int j = 0;

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

if (abs(P[i].x - midPoint.x) < d) {

strip[j] = P[i];

j++;

}

}

// Sort the points in the strip based on their y-coordinates

sort(strip, strip + j, compareY);

// Check for points in the strip that are closer than d

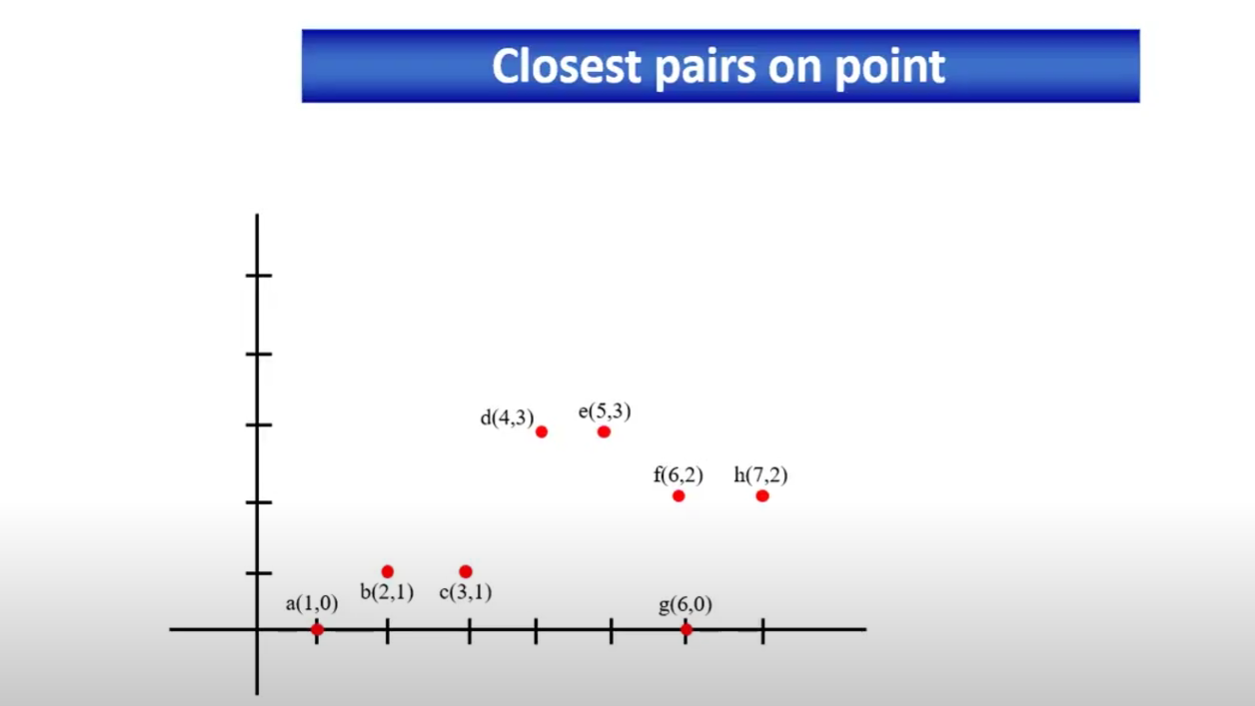
for (int i = 0; i < j; i++) {

for (int k = i + 1; k < j && (strip[k].y - strip[i].y) < d; k++) {

d = min(d, dist(strip[i], strip[k]));

}

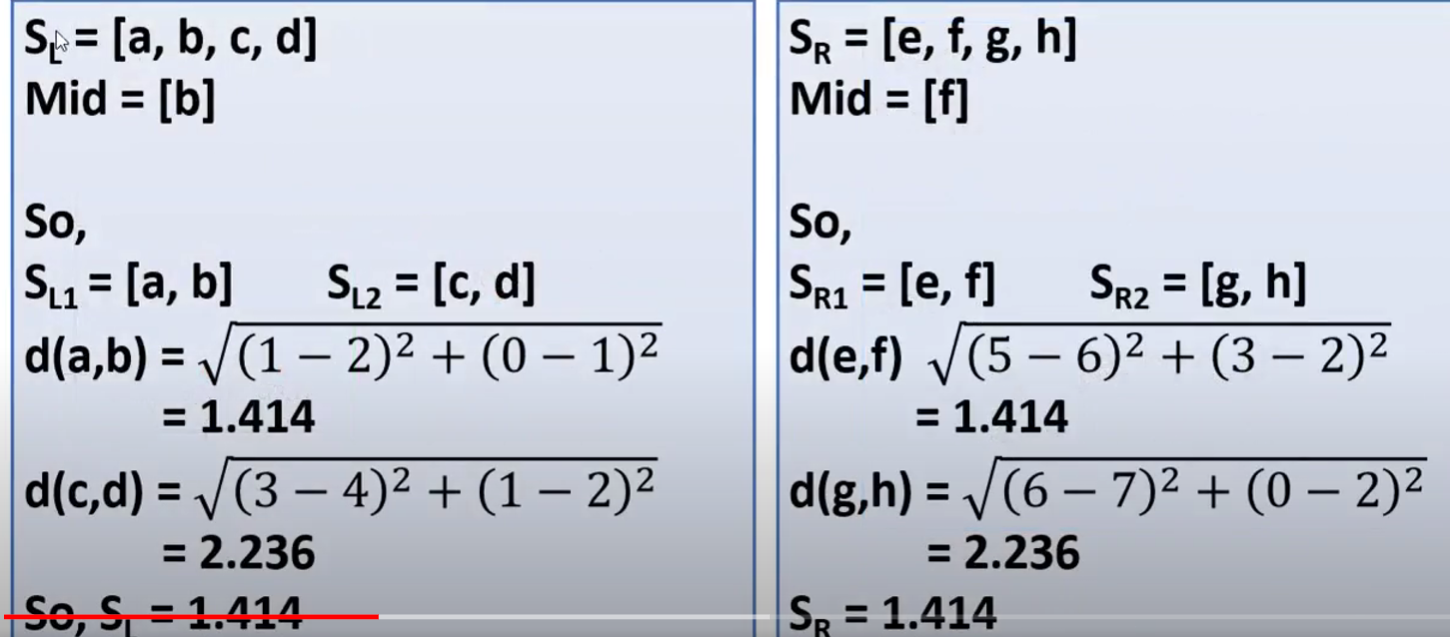
}

 A white background with black text

Description automatically generated

A graph of a function

Description automatically generated



A graph of a function

Description automatically generated

A math problem with numbers and symbols

Description automatically generated with medium confidence

So closest pair is (d,e) and distance is 1.