Julian Kliebhan, ID: 668484868
Advanced Algorithms and Data Structures
November 16, 2025

Project Description - Closest Pair of Points in 2D

Algorithmic Problem

We want to find an efficient algorithm that identifies a pair of points, $p, q \in P$, where P is a set of n points, that have the minimum Euclidean distance $d(p, q)$ among all possible pairs in the set.

Euclidean distance:

$$d(p, q) = \sqrt{\left(x_p - x_q\right)^2 + \left(y_p - y_q\right)^2}$$

We expect the algorithm to retun the minimum distance, δ, and the closest pair of points (p, q) from the given Set P.

A Naive Method and Its Running Time

The naiv method is to just compare all points with each other and return the minimum distance.

This simple brute force method has a running time of $O(n^2)$ since it has to perform $\sum_{i=1}^{n} i = \frac{n(n-1)}{2} = O(n^2)$ distance operations. The distance operations consists of doing a few arithmetic operations and a square root, taking constant time, $O(1)$.

The running time of the naive, brute-force method is inefficient and as we are about to see, we can find better algorithms to solve for this problem.

```
float min_dist(point_t points[]){
  float min = MAXFLOAT;

  for(int i=0; i<points.n-1; i++){
    point_t p = points[i];

    for(int k=i+1; k<points.n;k++){
      point_t q = points[k];

      float d = dist(p,q);
      if(min > d) min = d;
      }
    }
    return min;
}
```

Exploring better solutions

In the mid-1970, as the field of computational geometry was being formalized, researchers sought to find faster algorithms for fundamental problems. The $O(n \log n)$ algorithm presented by Shamos and Hoey was a major breakthrough. It's published in *"Shamos, M. I., & Hoey, D. (1975). 'Closest-point problems.'"* [1] In this paper it's proven that $\Omega(n \log n)$ is a lower bound for the closest pair problem. This is because the *element uniqueness problem* can be reduced to the *closest pair problem*. Therefore, the Shamos and Hoey algorithm is asymptotically optimal.

The Divide and Conquer Algorithm

This project focuses on the divide-and-conquer algorithm which is explained in different books. For example in *"Algorithm Design, Jon Kleinberg und Éva Tardos, Chapter 5.4: 'Finding the closest Pair of Points'"* [2]

The idea is simple:
1. Divide the set of points P recursivly into halves
2. Find the closest pair of each 'half'
3. Consider all not yet measured distances (this is the tricky part)
4. Take the minimum of all found candidates

Pseudocode

```
// Sort all points by x- and y- coordinate in O(nlogn)
let points_x = sort(points, points.x);
let points_y = sort(points, points.y);

// Recursivly search for the minimum distance
let dmin = closest_pair(points_x, points_y);

function closest_pair(points_x, points_y){
  n = points_x.length;

  // Base Case, each O(1)
  if(n == 2) return distance(points_x);
  if(n == 3) return min_dist_three_points(points_x) // returns the min of 3 points

  // Divide O(n)
  let xm = points_x[n/2];
  let lx = points_x[0..xm-1];
  let rx = points_x[xm..n-1];

  let ly, ry = [];
  for p in points_y:
    if(p.x <= xm.x) ly.push(p)
    else ry.push(p)

  //Conquer
  let dl = closest_pair(lx, ly);
  let dr = closest_pair(rx, ry);
  let dmin = min(dl, dr);

  //Combine O(n)
  let strip_y = create_strip(points_y, dmin, xm)  // contains all p in points_y
  // where abs(p.x - py[xm].x) < dmin

  let ds = dmin;
  for i in 0 .. (strip_y.n -1):
    for k in i+1 .. min(i+7, strip_y.n):
    // only need to check against the next 7 neighbors
      dist = distance(strip_y[i], strip_y[k]);
      ds = min(ds, dist);

  return min(dmin, ds);
}
```

A closer look at the algorithm

While the divide and conquer parts are pretty straight forward, the combine part does need a second look. At the point where we checked all recursivly created “halves”, we didn’t check points that were put in different sections, but still could be within a minimal distance. The clever part is to only check the next 7 neighbors in a 2δ wide y-strip around the median x-value. The fact that we only need to check a constant amount of neighbors is the real breakthrough of this algorithm. In the beginning it was calculated to at least check the next 15 neighbors, but over the time it was found to be true that 7 is enough. This can be shown with a precise geometric proof as seen in the book *“Introduction to algorithms, Chapter 33.4 ‘Finding the closest pair of points’”* [3].

Time Complexity

First we need to create two sorted lists. This takes $O(n \log n)$ time. Next we divide the lists, creating `lx` and `rx` takes $O(1)$ and reducing the points_y list to `ly` and `ry` takes $O(n)$ time. Now we make $2\times$ recursive calls with $T(\frac{n}{2})$. Creating the `strip_y` list takes another $O(n)$. The total recurrence is:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

By the Master Theorem, this recurrence relation solves to:

$$T(n) = O(n \log n)$$

With the $O(n \log n)$ from sorting we end up with: $O(n \log n)$ to find the closest pair of points.

A small example

1. Find `xm1`
2. Devide into Left and Right
3. Find `xmL2` and `xmR2` respectivly
4. Since all sets only contain 3 or less elements we can reporf the local minimum of each set: $\delta 1, \delta 2, \delta 3$ and $\delta 4$ and find the minimum on the way up:

   $$\min[\min(\delta 1, \delta 2), \min(\delta 3, \delta 4)] = \delta 2$$

5. For each `xm`-line we look at the $2\times\ \delta$ `y-strip`,
6. Finding the minimum distance in the $2\times\ \delta$ `y-strip` gives us the final candidate δS,
7. Comparing our last candidates results in $\min(\delta 2, \delta S) = \delta S$

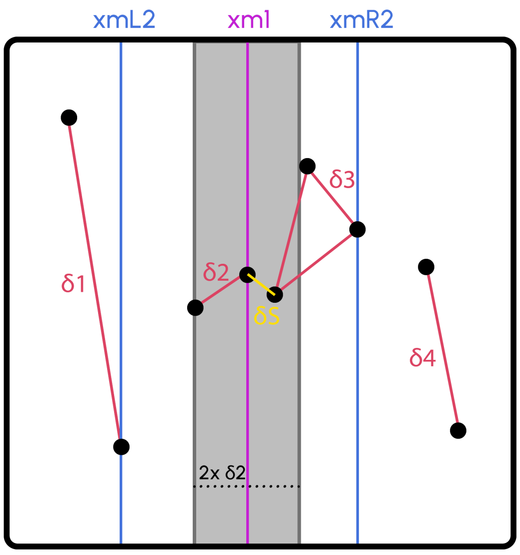

Figure 1: Closest Pair walktrough.
Only the last $2\times\ \delta$ `y-strip` highlighted.

Bibliography

[1] M. Shamos and D. Hoey, “CLOSEST-POINT PROBLEMS.” Accessed: Nov. 14, 2025. [Online]. Available: https://euro.ecom.cmu.edu/people/faculty/mshamos/1975ClosestPoint.pdf

[2] J. Kleinberg and É. Tardos, *Algorithm design*. Pearson, 2014. Accessed: Nov. 14, 2025. [Online]. Available: https://theswissbay.ch/pdf/Gentoomen%20Library/Algorithms/Algorithm%20Design%20-%20John%20Kleinberg%20-%20%C3%89va%20Tardos.pdf

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, and E. Al, *Introduction to algorithms*. Mit Press ; Boston [Etc, 2007. Accessed: Nov. 15, 2025. [Online]. Available: https://www.cs.mcgill.ca/~akroit/math/compsci/Cormen%20Introduction%20to%20Algorithms.pdf