Closest Pair Report

Adam Tovatt

May 9, 2025

1 Results

All test cases passed when running the provided `check_solution.sh` script. The program successfully solved even the largest test case with 1,000,000 points. On a modern machine, the total runtime for this case was around 1.4 seconds. The majority of the time is spent sorting the input points and computing the recursive closest pair distance. For example, for the largest input, the time spent on different parts of the solution is as follows:

Section	**Time (ms)**	**Percent of Total**
readInput	250	18.0%
sortPoints	491	35.2%
computeClosestDistance	655	46.9%
fullSolve	1401	100.2%

2 Implementation details

I implemented a standard divide-and-conquer algorithm to solve the Closest Pair of Points problem. The input consists of up to 1,000,000 2D points, and the goal is to compute the smallest Euclidean distance between any two of them. The core steps are:

- Sort the points by x-coordinate and y-coordinate at the beginning.
- Recursively divide the points into left and right halves.
- Compute the closest distance in each half.
- Construct a vertical strip around the midpoint and check potential cross-pair distances between the halves.

The main data structure used is a list of `Point` structs, and standard sorting algorithms are used to sort by x and y. The strip step only checks each point against at most 7 other nearby points thanks to geometric constraints.

The overall time complexity is $O(N \log N)$, where N is the number of points. This comes from the recursive division (like mergesort) and a merge step that does linear work per level.

Discussion Questions

What is the time complexity, and why?
The time complexity is $O(N \log N)$. This is because the algorithm splits the data recursively (which takes $\log N$ steps), and at each step it processes all N points by scanning the strip or combining results. Sorting the input initially also takes $O(N \log N)$, and together this gives the final complexity.

Why is it sufficient to check a few points along the mid line?
When checking for possible closest pairs that cross the dividing line, we only look at points within a narrow vertical strip of width 2δ, where δ is the current best distance. Due to geometric packing limits in 2D, it's been proven that you only need to compare each point with at most 7 others in the strip (sorted by y-coordinate). Any additional points would be too far vertically to be closer than δ.

Draw a picture and show/describe when each distance is checked in your solution!
In the recursive step, distances are checked between all pairs in the left half and in the right half. Then, in the "combine" step, distances are checked between points that lie near the dividing line (i.e., within δ in the x-direction), and for those, only 7 ahead in the y-sorted list are checked. So each distance is checked either during brute-force (base case) or in the merge step when the point falls within the strip. *(See the figures on the next page)*

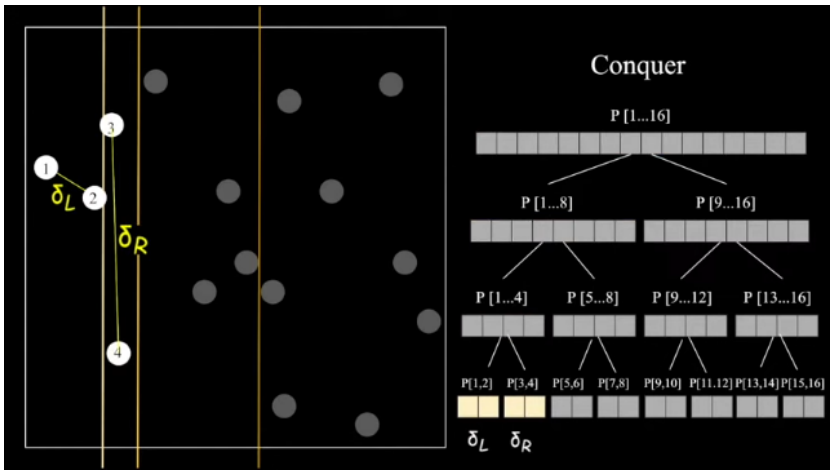

Figure 1: Recursive case distance comparisson illustration. Taken from this youtube-video by Linq Qi also known as ""iDeer70".

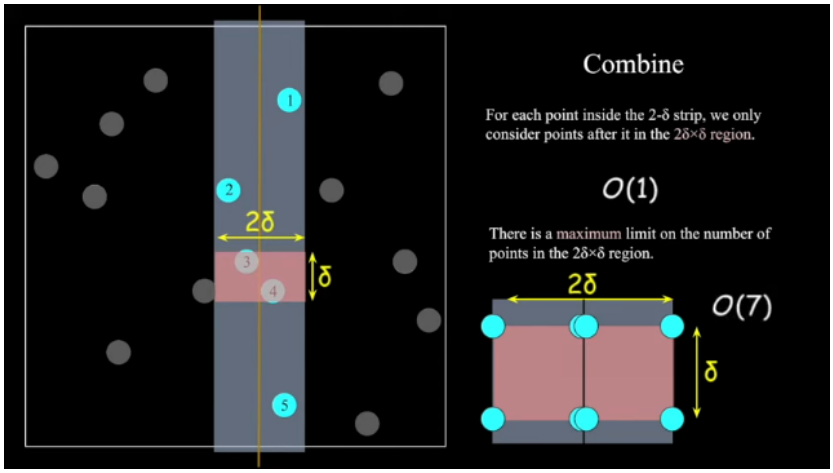

Figure 2: Illustration of the strip search performed to ensure no closest pairs are missed due to the splitting. Taken from this youtube-video by Linq Qi also known as ""iDeer70".

When do you break the recursion and use brute force?
The recursion breaks when the number of points in the subproblem is 3 or fewer. At that point, the function simply compares all pairs directly with brute force since the overhead of recursive division is not worth it for such small input.