Yusuf Fawzy Elnady

Closest Pair Algorithm (Divide and Conquer)

Summary:

Given an array of **n points** in the plane, the problem is to **find out the closest pair of points** in the array using the Euclidean distance formula:

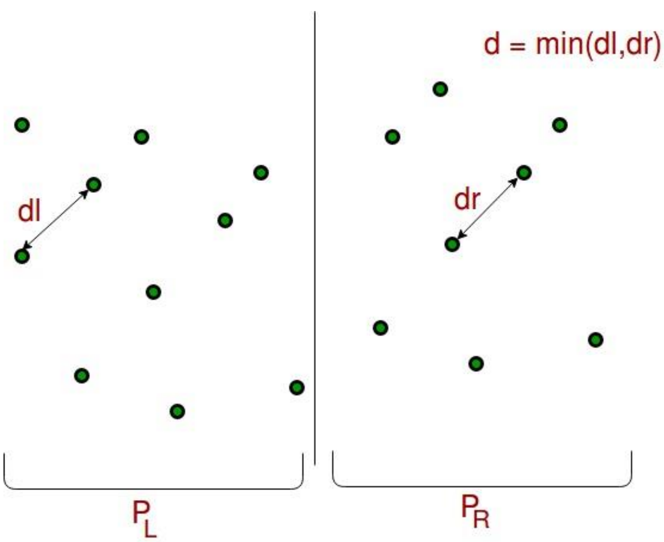
```
d = sqrt((px - qx)^2 + (py - qy)^2)
```

The Brute force solution is **O(n^2),** compute the distance between each pair and return the smallest. However, but using divide and conquer strategy, it can be computed in **O(n (Logn)^2).**

The project is built using **C++, Code::Blocks.**

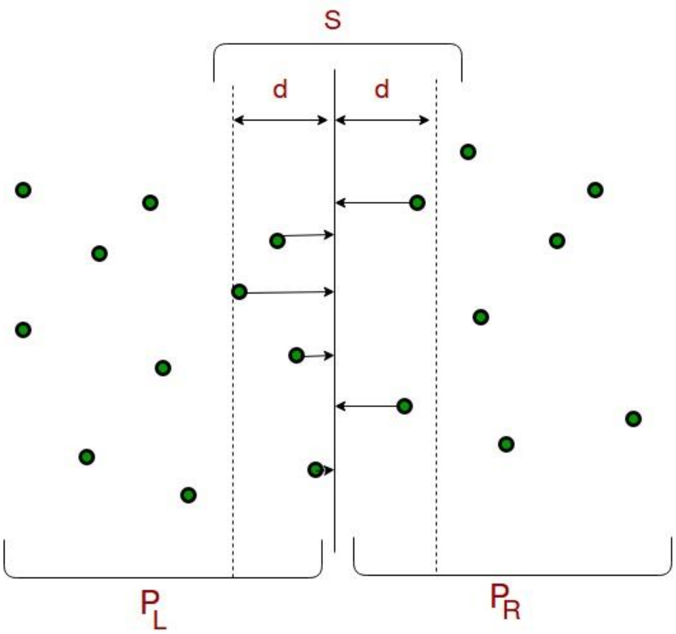
Algorithm:

As a **pre-processing step**, the input array will be **sorted** according to **x coordinates**.
1. **Divide** the given array into **two halves**. The **first subarray** contains points from P[0] to P[(n/2) - 1], and the **second subarray** contains points from P[n/2] to P[n-1].
2. **Recursively** find the **smallest distances** in both subarrays. Let the distances be **dl** and **dr**. Find the minimum of **dl** and **dr**. Let the minimum be **“d”**.



3. From the above 2 steps, we have an **upper bound d of minimum distance**. Now we need to consider the pairs such that **one point** in pair is **from left half** and **other** is **from right half**. Consider the vertical line passing in the middle between **P[(n/2) - 1]** and **p[n/2]**, we find all points whose X coordinate is **closer than d to the middle vertical line**. Build an array **strip[]** of all such points.

Yusuf Fawzy Elnady



4. **Sort** the array strip[] according to **Y coordinates**. This step is **O(nLogn).**
5. **Find** the **smallest** distance in **strip[]**. This is tricky. From first look, it seems to be a **O(n^2) step**, but it is actually **O(n).** It can be proved **geometrically** that for every point in strip, we only need to check at most 7 points after it (note that strip is sorted according to **Y coordinates**).
6. Finally, **return** the **minimum** of “d” and **distance** calculated in above step (step 5)

Why do we only need to check at most 7 points in the strip?

Hypothesis: Let the current point is **qi**, let the min distance till now be **k**. There are **at most 7 points qj** such that **(yi - yj)** < (**min distance so far 'k'**).

Proof: Each such **qj** must lie either in the left or in the right **(k * k) square.** Within each square (the left square or the right square), all points have distance **k** from each other, because they are in the same half, and the min distance in the same half is found by recursion to be k. We can pack at most 4 such points into one square, so we have 8 points total (**including qi**).