CS340 Analysis of Algorithms

Handout:	**6**	**Professor:**	**Dianna Xu**
Title:	**Closest Pair**	**E-mail:**	`dxu@cs.brynmawr.edu`
Date:		**URL:**	**http://cs.brynmawr.edu/cs340**

Sample description, pseudo code and proof of correctness for closest pair divide-and-conquer algorithm. Please refer to lecture notes for details of the algorithm design and time analysis. Recall that given a set of points P of size n, the algorithm finds the closest pair $p,q \in P$ and outputs the Euclidean distance $\|pq\|$.

1 Description

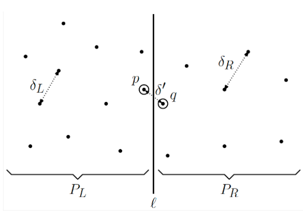
We begin by presorting the points, both with respect to their x- and y-coordinates. Let P_x denote the points of P sorted by x, and let P_y be the points of P sorted by y. We can compute these sorted arrays in $O(nlogn)$ time. Note that this initial sorting is done only once. In particular, the recursive calls do not repeat the sorting process.

Base case: If $|P| \leq 3$, then just solve the problem by brute force in O(1) time.

Divide: Otherwise, partition the points into two subarrays P_L and P_R based on their x-coordinates. In particular, imagine a vertical line l that splits the points roughly in half. In the same way that we represented P using two sorted arrays, we do the same for P_L and P_R. Since we have presorted P_x by x-coordinates, we can determine the median element for l in constant time. After this, we can partition each of arrays P_x and P_y in O(n) time each.

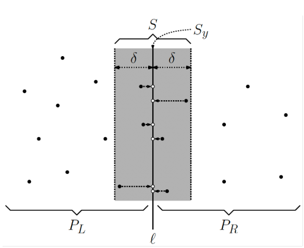
Conquer: Compute the closest pair within each of P_L and P_R, by invoking the algorithm recursively. Let δ_L and δ_R be the closest pair distances in each case. Let $\delta = min(\delta_L, \delta_R)$.

Combine: Note that δ is not necessarily the final answer, because there may be two points



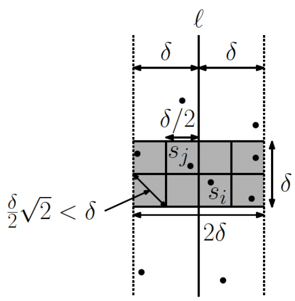
that are very close to one another but are on opposite sides of l and not in the same half. To complete the algorithm, we want to determine the closest pair of points between the sets, that is, the closest points $p \in P_L$ and $q \in P_R$. Since we already have an upper bound δ on the closest pair, it suffices to solve the following restricted problem: find $(p,q), p \in P_L, q \in P_R$ and $\|pq\| < \delta$, then we will return its distance δ' (or ∞ if none exists). We return $min(\delta, \delta')$ as the final result.

Closest Pair Between the havles:



Observe that if such a pair of points exists, we may assume that both points lie within distance δ of l, for otherwise the resulting distance would exceed δ. Let S denote this subset of P that lies within a vertical strip of width 2δ centered about l. Let $S_y = (s_1, ..., s_m)$ denote the points of S sorted by their y-coordinates. The key observation is that if S_y contains two points that are within distance δ of each other, these two points must be within a constant number of positions of each other in the sorted array S_y and therefore can be found with a simple linear search.

Lemma 1.1. *Given any two points s_i, s_j in S_y, if dist(s_i, s_j) $< \delta$, then $|j-i| \leq 7$.*



Proof: Suppose that dist(s_i, s_j) $<\delta$. Since they are in S, they are each within distance δ of l. The y-coordinates of these two points can differ by at most δ as well. So they must both reside in a rectangle of width 2δ and height δ centered about l. Split this rectangle into eight identical squares each of side length $\frac{\delta}{2}$. A square of side length x has a diagonal of length $x\sqrt{2}$, and no two points within such a square can be farther away than this.

Therefore, the distance between any two points lying within one of these eight squares is at most $\frac{\delta\sqrt{2}}{2} = \frac{\delta}{\sqrt{2}} < \delta$. Since each square lies entirely on one side of l, no square can contain two or more points of P, since otherwise, these two points would contradict the fact that δ is the closest pair seen so far. Thus, there can be at most eight points of S in this rectangle, one for each square. Therefore, $|j-i| \leq 7$.

Lemma 1.2. *By Lemma 1.1, a pair $(p,q), p\in P_L, q\in P_R$, such that $\|pq\| < \delta$, if one exists, can be found by linear search in S_y, in 8-point blocks.*

2 Pseudocode

```
// presort P by x and y into P_x and P_y
Function closestPair(P=(P_x,P_y))
    if |P| ≤ 3 then
        solve by brute force
    end
    else
        split P into two halves P_L and P_R based on their x coordinates (using P_x, via the vertical
         line l)
        δ_L = closestPair(P_L)
        δ_R = closestPair(P_R)
        δ=min(δ_L,δ_R)
        for i = 1 to n do
            if P_y[i] is within δ of l then
                append P_y[i] to S_y
            end
        end
        δ' = findClosest(S_y)
    end
    return min(δ,δ')
Function findClosest(S_y)
    m = |S_y|
    δ'=∞
    for i = 1 to m do
        for j=i+1 to min(m,i+7) do
            if dist(S_y[i], S_y[j]) < δ' then
                δ' = dist(S_y[i], S_y[j])
            end
        end
        return δ'
    end
```

3 Correctness Proof

We prove the correctness of `closestPair` with structral induction on the length of P.

- Base case: $|P| \leq 3$, brute force works
- IH: assume that `closestPair` works correctly $\forall |P| \leq n-1$.
- Want to show: `closestPair` returns the correct δ for $|P| = n$. We have three cases. The closest pair either

  1. has both points in P_L, in which case $\delta = \delta_L$
  2. has both points in P_R, in which case $\delta = \delta_R$
  3. has one in each, which means both are in S_y, in which case $\delta = \delta'$

  The IH covers both recursive calls (because they are called on at most half of the length of P and therefore strictly less than n). It follows that δ_L and δ_R are correct. By lemma 1.1 and 1.2, the `for` loop correctly constructs S_y and `findClosest` correctly finds and returns δ'. The *min* comparisons will correctly identify the minimum of δ_L, δ_R and δ' and return as final δ.