Design and Analysis of ALGORITHMS [21AIE212]

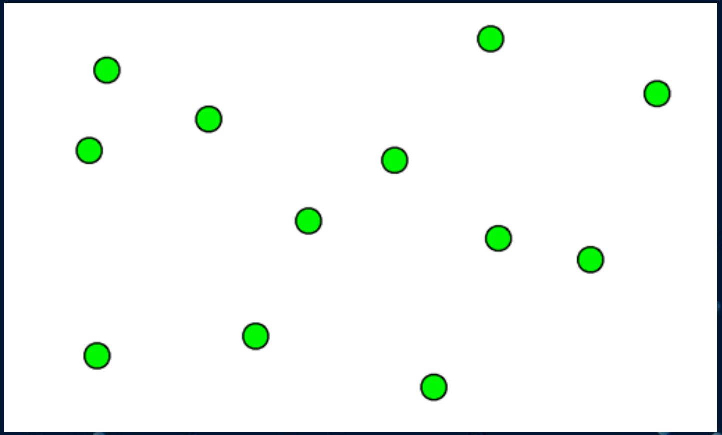
Submitted by: Vikhyat Bansal
Roll Number: [CB.EN.U4AIE21076]

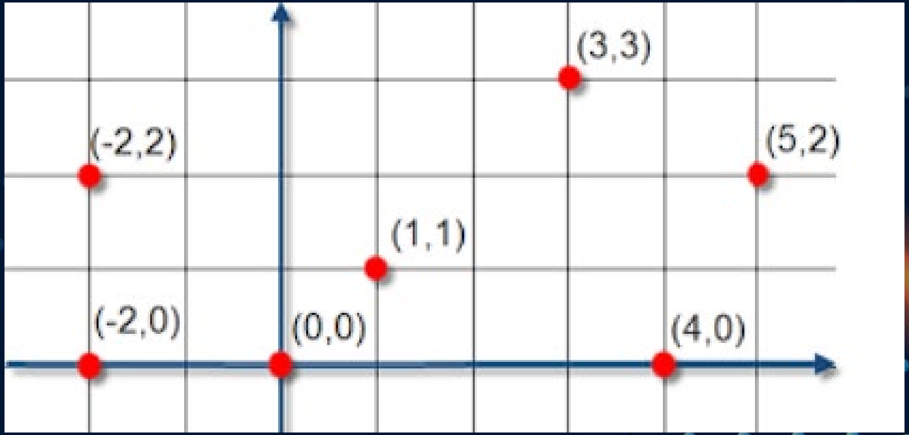
In-Class Assignment – 1

Problem: Find the closest pair of points

Given a set of points {p1, . . . , pn} find the pair of points {pi , pj} that are closest together.

These algorithms formed the foundations of the then-fledgling field of *computational geometry*, and they have found their way into areas such as graphics, computer vision, geographic information systems, and molecular modelling.
One more touch and go example of this problem can be airplane distance detection to avoid collisions between them when enroute.





ASSUMPTIONS & GRIEVANCEAS

- We will assume that no two points in *P* have the same *x*-coordinate or the same *y*-coordinate.

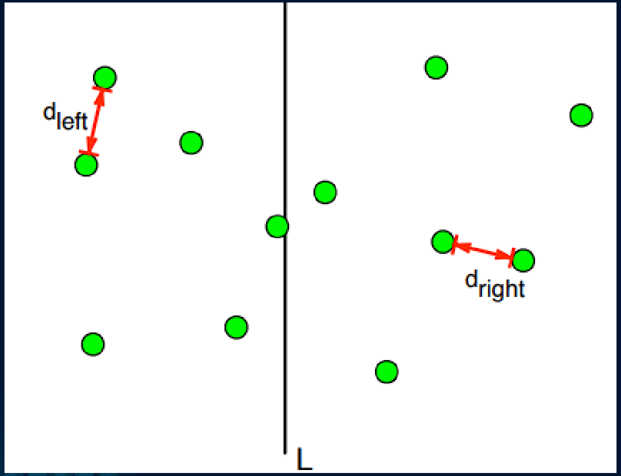
- It's instructive to consider the one-dimensional version of this problem for a minute, since it is much simpler and the contrasts are revealing. We'd first sort them, in *O(n* log *n)* time, and then we'd walk through the sorted list, computing the distance from each point to the one that comes after it.

- In two dimensions, we could try sorting the points by their *y*-coordinate (or *x*-coordinate) and hoping that the two closest points were near one another in the order of this sorted list. But it is easy to construct examples in which they are very far apart, preventing us from adapting our one-dimensional approach.

APPROACH:
DIVIDE AND CONQUER

DIVIDE

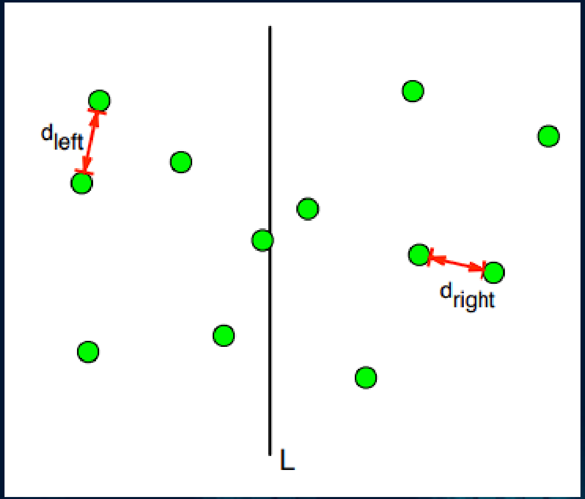
Split the points with line L so that half the points are on each side.
Recursively find the pair of points closest in each half.

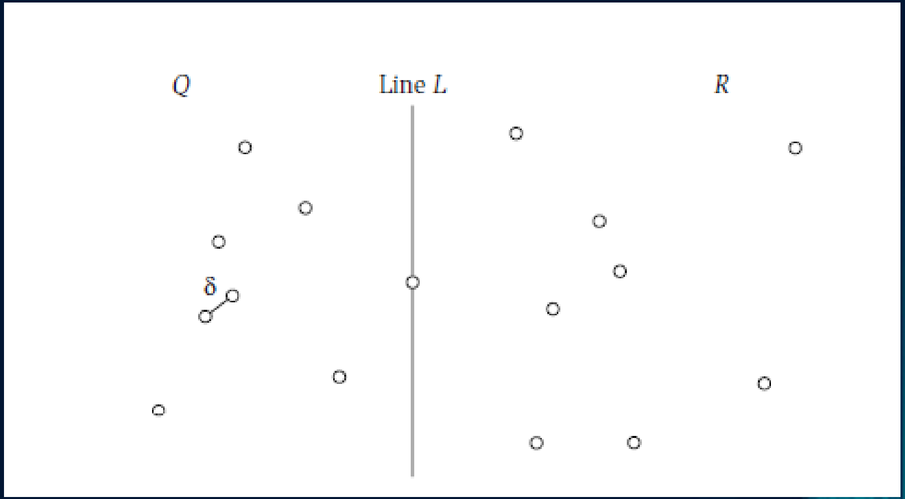


APPROACH: MERGE-SORT (Similar)

MERGE

Let d = min{d_{left}, d_{right}}.
Let *x*∗ denote the *x*-coordinate of the rightmost point in *Q*, and let *L* denote the vertical line described by the equation *x* = *x*∗. This line *L* "separates" *left* from right.
d would be the answer, except maybe L split a close pair!



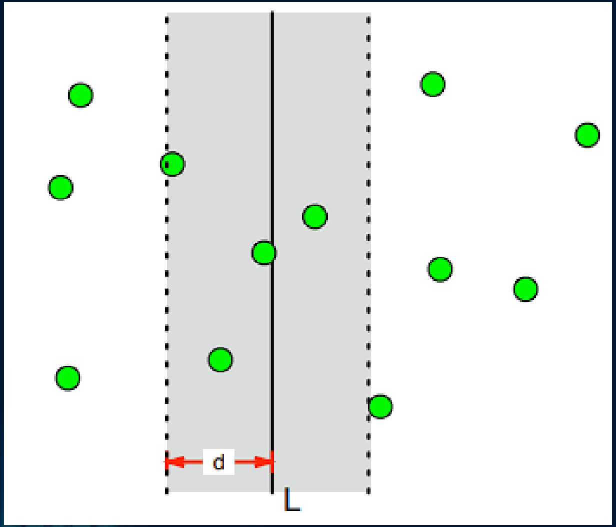


APPROACH: MERGE-SORT (Similar)

NARROW BAND NEAR L

If there is a pair $\{p_i , p_j\}$ with $\text{dist}(p_i , p_j) < d$ that is split by the line, then both p_i and p_j must be within distance d of L.

Let S_y be an array of the points in that region, sorted by increasing y-coordinate value.



APPROACH: MERGE-SORT (Similar)

ARRAY MIGHT CONTAIN ALL POINTS

- Let S_y be an array of the points in that region, sorted by increasing y-coordinate value.
- S_y might contain all the points, so we can't just check every pair inside it.

Theorem:

Suppose $S_y = p_1, \ldots, p_m$. If $\text{dist}(p_i, p_j) < d$ then $j - i \leq 15$.

In other words, if two points in S_y are close enough in the plane, they are close in the array S_y

APPROACH: DIVIDE & CONQUER

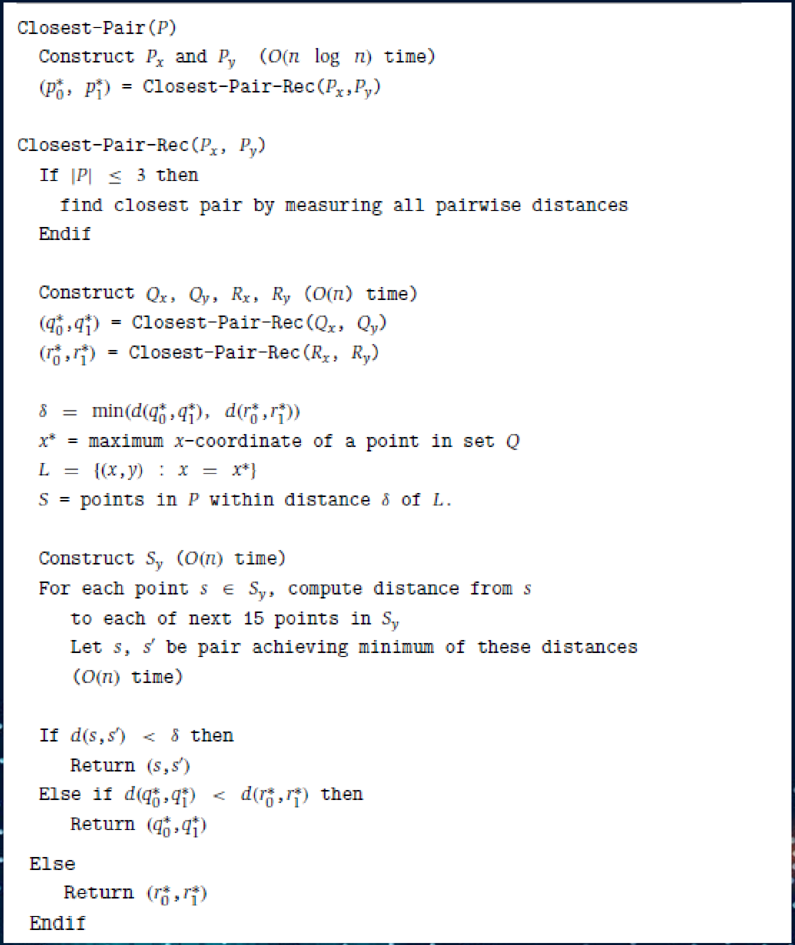
COMBINING

We can conclude the algorithm as follows.
We make one pass through *Sy*, and for each *s* ∈ *Sy*, we compute its distance to each of the next 15 points in *Sy*.
We will have computed the distance of each pair of points in *S* (if any) that are at distance less than *d* from each other. So having done this, we can compare the smallest such distance to *d*, and we can report one of two things:

(i) the closest pair of points in *S*, if their distance is less than *d*;
(ii) the (correct) conclusion that no pairs of points in *S* are within *d* of each other.

In case (i), this pair is the closest pair in *P*;
in case (ii), the closest pair found by our recursive calls is the closest pair in *P*.

PSEUDO-CODE



```
Closest-Pair($P$)
  Construct $P_x$ and $P_y$  ($O(n \log n)$ time)
  ($p_0^*$, $p_1^*$) = Closest-Pair-Rec($P_x$,$P_y$)

Closest-Pair-Rec($P_x$, $P_y$)
  If $|P| \leq 3$ then
    find closest pair by measuring all pairwise distances
  Endif

  Construct $Q_x$, $Q_y$, $R_x$, $R_y$ ($O(n)$ time)
  ($q_0^*$,$q_1^*$) = Closest-Pair-Rec($Q_x$, $Q_y$)
  ($r_0^*$,$r_1^*$) = Closest-Pair-Rec($R_x$, $R_y$)

  $\delta = \min(d(q_0^*,q_1^*), d(r_0^*,r_1^*))$
  $x^*$ = maximum $x$-coordinate of a point in set $Q$
  $L = \{(x,y) : x = x^*\}$
  $S$ = points in $P$ within distance $\delta$ of $L$.

  Construct $S_y$ ($O(n)$ time)
  For each point $s \in S_y$, compute distance from $s$
     to each of next 15 points in $S_y$
     Let $s$, $s'$ be pair achieving minimum of these distances
     ($O(n)$ time)

  If $d(s,s') < \delta$ then
     Return $(s,s')$
  Else if $d(q_0^*,q_1^*) < d(r_0^*,r_1^*)$ then
     Return $(q_0^*,q_1^*)$
 Else
    Return $(r_0^*,r_1^*)$
 Endif
```

Example

Consider the following set of points in a plane: (1, 3), (3, 1), (4, 6), (7, 8), (9, 2), (10, 5)

The algorithm first sorts the points by their x-coordinates:
(1, 3), (3, 1), (4, 6), (7, 8), (9, 2), (10,5) .

It then recursively splits the set of points into left and right halves based on the last x-coordinate of one half, which is 5:

left half: (1, 3), (3, 1), (4, 6)

right half: (7, 8), (9, 2), (10, 5)

Example

The algorithm then recursively computes the closest pair of points on the left and right halves. On the left half, the closest pair of points is (1, 3) and (3, 1) with distance 2.828. On the right half, the closest pair of points is (7, 8) and (9, 2) with distance 6.325.

The algorithm takes the minimum distance between the two pairs, which is 2.828, and creates a band of points that are within this distance of the median line:

Band: (4, 6), (7, 8), (9, 2)

Example

The algorithm then sorts the points in the strip by their y-coordinates and checks for the closest pair of points.

The closest pair within the strip is (4, 6) and (7, 8) with distance 3.6055

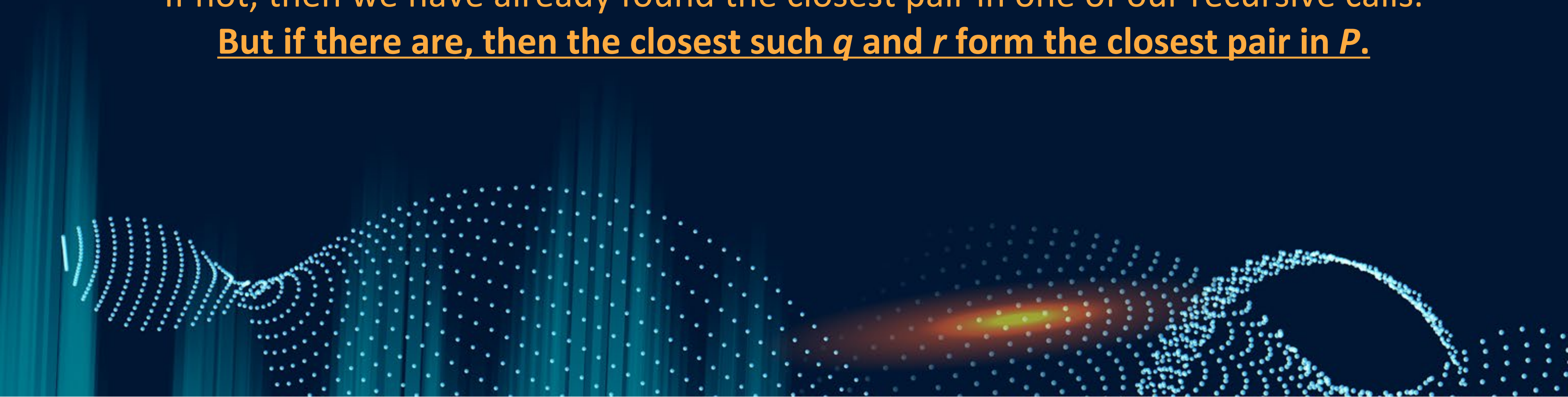
Since the distance between the closest pair within the band is more than the minimum distance between the left and right halves, the algorithm returns the closest pair within the strip, which is (1,3) and (3, 1) with distance 2.828.

PROOF OF CORRECTNESS

Let δ be the minimum of $d(q\ast 0,q\ast 1)$ and $d(r\ast 0\ ,r\ast 1\ )$.

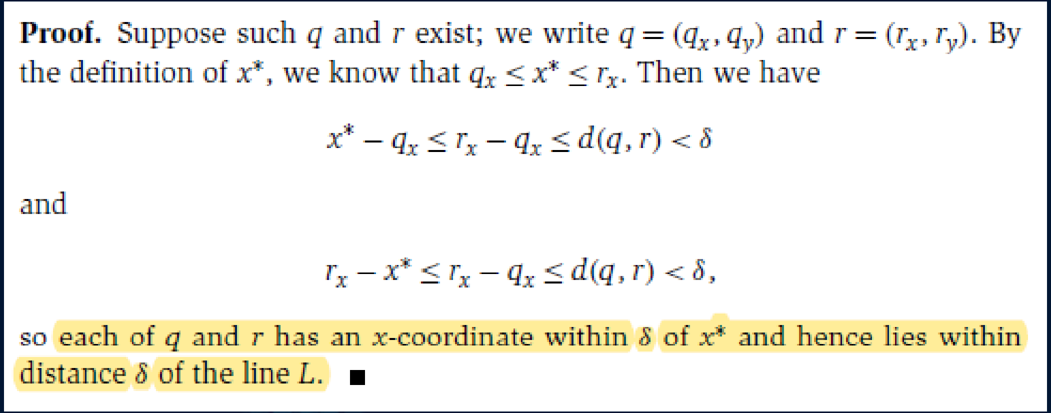
The real question is: Are there points $q \in Q$ and $r \in R$ for which $d(q, r) < d$?

If not, then we have already found the closest pair in one of our recursive calls.
<u>**But if there are, then the closest such *q* and *r* form the closest pair in *P*.**</u>



PROOF OF CORRECTNESS

If there exists $q \in Q$ and $r \in R$ for which $d(q, r) < \delta$, then each of q and r lies within a distance δ of L.



Proof. Suppose such q and r exist; we write $q = (q_x, q_y)$ and $r = (r_x, r_y)$. By the definition of x^*, we know that $q_x \le x^* \le r_x$. Then we have

$$x^* - q_x \le r_x - q_x \le d(q, r) < \delta$$

and

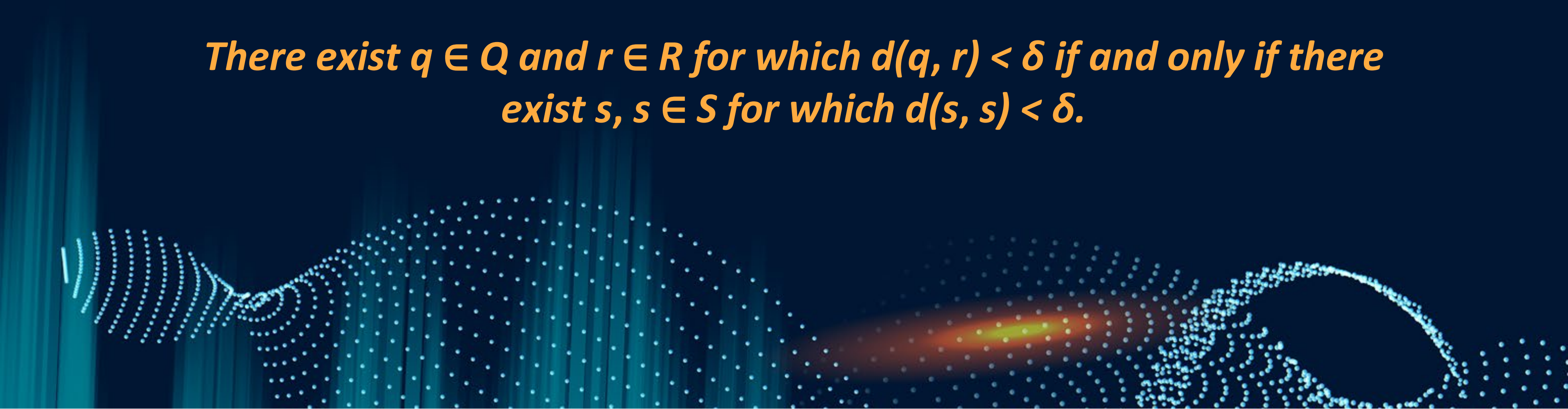
$$r_x - x^* \le r_x - q_x \le d(q, r) < \delta,$$

so each of q and r has an x-coordinate within δ of x^* and hence lies within distance δ of the line L. ■

PROOF OF CORRECTNESS

Let $S \subseteq P$ denote this
set, and let S_y denote the list consisting of the points in S sorted by increasing y-coordinate. By a single pass through the list P_y, we can construct S_y in $O(n)$ time.

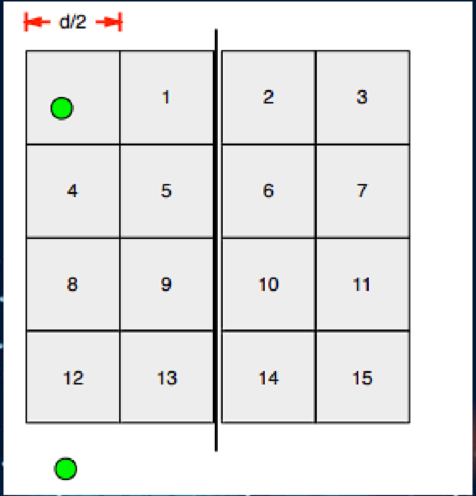
There exist $q \in Q$ and $r \in R$ for which $d(q, r) < \delta$ if and only if there exist $s, s \in S$ for which $d(s, s) < \delta$.



PROOF OF CORRECTNESS

If s, s’ ∈ S have the property that d(s, s) < δ, then s and s are within 15 positions of each other in the sorted list Sy.

- Then, at least 3 full rows separate them (the packing shown is the smallest possible).
- But the height of 3 rows is > 3d/2, which is > d.
- So the two points are farther than d.



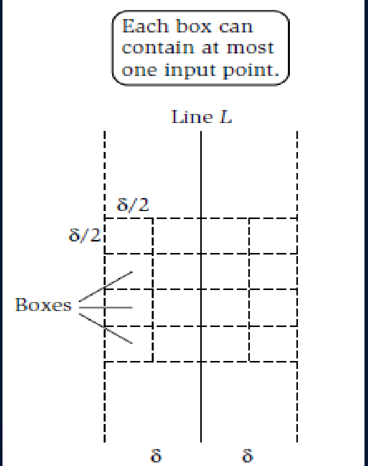


Figure 5.7 The portion of the plane close to the dividing line L, as analyzed in the proof of (5.10).

PROOF OF CORRECTNESS

We prove the correctness by induction on the size of P, the case of $|P| \leq 3$ being clear.

For a given P, the closest pair in the recursive calls is computed correctly by induction. Remainder of the algorithm correctly determines whether any pair of points in S is at distance less than δ, and if so returns the closest such pair. Now the closest pair in P either has both elements in one of Q or R, or it has one element in each. In the former case, the closest pair is correctly found by the recursive call; in the latter case, this pair is at distance less than δ, and it is correctly found by the remainder of the algorithm.

RUNTIME

Total Running Time:

- Divide set of points in half each time: $O(\log n)$ depth recursion
- Merge takes $O(n)$ time.
- Recurrence: $T(n) \leq 2T(n/2) + cn$
- Same as Merge-Sort $\Longrightarrow$ $O(n \log n)$ time.

REFERENCES

- https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/closepoints.pdf
- Book: - Algorithm Design (By Tardos)

THANK YOU!!