

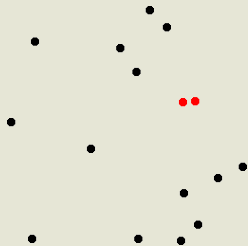
Closest Pair

by Divide & Conquer

$$S = \{p_1, p_2, \dots, p_n\} \subset \mathbb{R}^2$$

WANTED: $p_i, p_j \in S$ such that

$$\text{dist}(p_i, p_j) = \delta = \min\{\text{dist}(p, q) \mid p, q \in S, p \neq q\}$$



Euclidean distance of $p = (p_x, p_y)$ and $q = (q_x, q_y)$:

$$\text{dist}(p, q) := \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Brute-force algorithm solves the problem in time $\Theta(n^2)$:

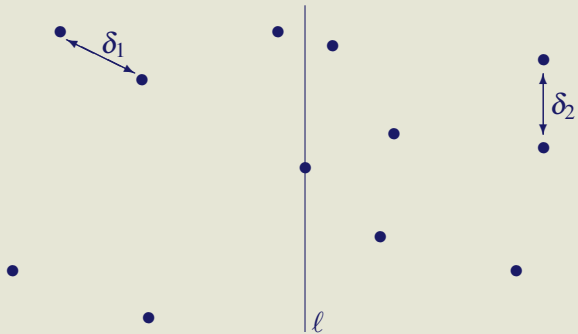
```
1   $\delta \leftarrow \infty$ 
2  for each  $p \in S$ 
3      do for each  $q \in S$ 
4          do if ( $p \neq q$ ) and  $\text{dist}(p, q) < \delta$ 
5              then  $\delta = \text{dist}(p, q)$ 
6                   $\text{closest\_pair} \leftarrow (p, q)$ 
7  return  $\text{closest\_pair}$ 
```

Sort points lexicographically by (x,y) -coordinates in increasing order and store the points in array $S[1..n]$.

Algorithm sketch:

$\text{MINDIST}(S, i, j)$

```
1  if  $i = j$ 
2      then return  $\infty$ 
3      else  $k \leftarrow \lfloor (i + j) / 2 \rfloor$ 
4           $\delta_1 \leftarrow \text{MINDIST}(S, i, k)$ 
5           $\delta_2 \leftarrow \text{MINDIST}(S, k + 1, j)$ 
6          MERGE
7          return  $\delta_0$ 
```



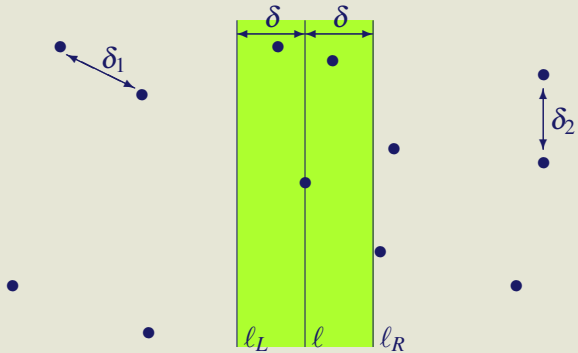
Observations:

- $\delta_0 \leq \delta = \min(\delta_1, \delta_2)$
- If there are $\alpha < \beta$ such that

$$\text{dist}(S[\alpha], S[\beta]) < \delta$$

then $i \leq \alpha \leq k$ and $k < \beta \leq j$.

- let ℓ_L be the vertical line at distance δ left of the separating line ℓ and let ℓ_R be the vertical line at distance δ right of ℓ . Then $S[\alpha]$ lies in the strip bounded by ℓ_L and ℓ and $S[\beta]$ lies in the strip bounded by ℓ and ℓ_R .



We can ignore all points outside of these strips.

Step 1:

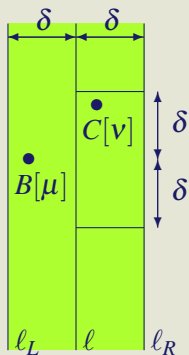
We copy the relevant points into separate containers B and C :

```
1   $\delta \leftarrow \min(\delta_1, \delta_2)$ 
2   $a \leftarrow 1$ ;
3  for  $l \leftarrow i$  to  $k$ 
4      do if  $S[l]_x > S[k]_x - \delta$ 
5          then  $B[a] \leftarrow S[l]$ 
6               $a \leftarrow a + 1$ 
7   $a \leftarrow 1$ 
8  for  $l \leftarrow k + 1$  to  $j$ 
9      do if  $S[l]_x < S[k]_x + \delta$ 
10     then  $C[a] \leftarrow S[l]$ 
11          $a \leftarrow a + 1$ 
```

Observations:

- if $\delta_0 < \delta$ then there are μ, ν such that $\text{dist}(B[\mu], C[\nu]) < \delta$ and

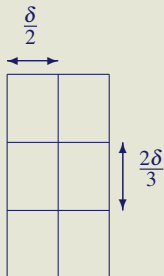
$$B[\mu]_y - \delta < C[\nu]_y < B[\mu]_y + \delta$$



Lemma

For each $B[l]$ there are at most 6 points p in C such that

$$B[l]_y - \delta < p_y < B[l]_y + \delta$$



A sub-box has diagonal $\frac{5\delta}{6} < \delta$ and contains at most one point!

Step 2:

We sort the points in B and C by y -coordinate in increasing order.

Step 3:

```

1   $j \leftarrow 1$ 
2  for  $i \leftarrow 1$  to  $|B|$ 
3      do while  $C[j]_y \leq B[i]_y - \delta$ 
4          do  $j \leftarrow j + 1$ 
5           $l \leftarrow j$ 
6          while  $C[l]_y < B[i]_y + \delta$ 
7              do  $\delta \leftarrow \min(\delta, \text{dist}(B[i], C[l]))$ 
8                   $l \leftarrow l + 1$ 
9  return  $\delta$ 

```

running time of step 3: $O(|B| + |C|)$

Let $T(n)$ be the worst-case running time of the algorithm described above.

\exists constants c, c' such that

$$T(n) \leq \begin{cases} c & n = 1 \\ c'n \log n + T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) & n > 1 \end{cases}$$

$$T(n) = O(n(\log n)^2)$$

Improved algorithm:

Combine the algorithm described above with MERGE-SORT:
After $\text{MINDIST}(S, i, j)$ has been executed, the points in $S[i..j]$ are sorted according to y -coordinates.

If points in $S[i..k]$ and $S[k+1..j]$ are already sorted according to y -coordinates, $S[i..j]$ can be sorted in time $O(j-i+1)$.

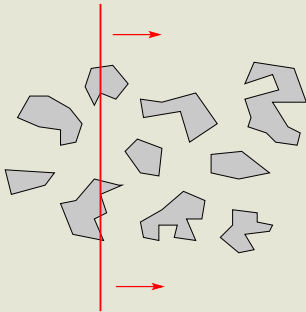
Thus we get a linear time bound for our MERGE step.

Theorem

Using divide & conquer, we can compute the minimum distance among a set of n points in the plane in time $O(n \log n)$. The same bound holds for the computation of the closest pair.

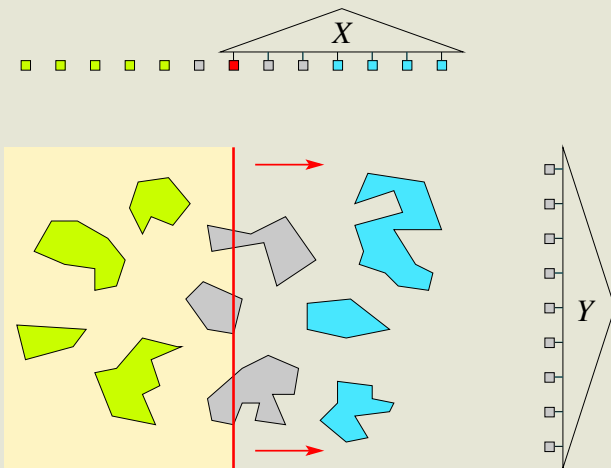
Plane Sweep

While the plane is (conceptually) swept by a straight line, processing geometric objects is started one by one as the objects are encountered by the sweep line and a partial solution to the given problem is maintained.



- Plane sweep is kind of an incremental construction. However, processing order is determined by the relative position of the geometric objects in the plane. Moreover, processing an object is not necessarily completed before processing another object starts.
- Plane sweep uses locality information, e.g. it uses knowledge on relative position of objects to detect and avoid irrelevant computation.

- objects not yet reached by the sweep line are called *sleeping*
- objects completely passed by the sweep line are called *dead*
- objects intersected by the sweep line are called *active*
- X-structure maintains discrete events, called *event points* or *transition points*, where “the status of the sweep line with respect to the objects” changes.
- Y-structure maintains information on the interaction of the sweep line with the active objects.



High-level description of a typical plane sweep

- 1 compute an initial solution \mathcal{S}
- 2 compute the set of event points known in advance and insert them into the X-structure \mathcal{X}
- 3 compute initial Y-structure \mathcal{Y}
- 4 **while** (\mathcal{X} is not empty)
- 5 **do** $x \leftarrow \mathcal{X}.delmin()$
- 6 process x and update \mathcal{S} , \mathcal{X} , and \mathcal{Y}

Pareto-optimal Points

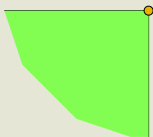
by Plane Sweep

$$S = \{p_1, p_2, \dots, p_n\} \subset \mathbb{R}^2$$

Definition

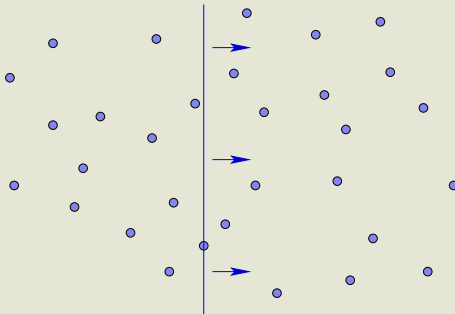
A point $p \in S$ is called *minimal* if

$$\{q \in S - \{p\} \mid q_x \leq p_x \text{ and } q_y \leq p_y\} = \emptyset$$



Problem (Minimal Points)

Given a finite set S of points in the plane, compute the sequence of minimal points in S sorted by x -coordinate in increasing order.



Observations:

- Let $p \in S$. Points in S right of p do not matter.
- It suffices to consider the point with smallest y -coordinate among the points left of p .

Sort points lexicographically by (x,y) -coordinates in increasing order and store the points in array $S[1..n]$.

SWEEP_MINIMA(S, n)

```
1  Report  $S[1]$ 
2   $q \leftarrow S[1]$ 
3  for  $i \leftarrow 2$  to  $n$ 
4      do if  $S[i]_y < q_y$ 
5          then report  $S[i]$ 
6           $q \leftarrow S[i]$ 
```

$\text{SWEEP_MINIMA}(S, n)$ correctly solves the Minimal Points Problem.

The worst-case running time $T(n)$ of $\text{SWEEP_MINIMA}(S, n)$ for already sorted elements is $O(n)$.

Initial sorting takes time $O(n \log n)$.

Theorem

Using plane sweep, we can compute the minimal points in a set of n points in the plane in time $O(n \log n)$.