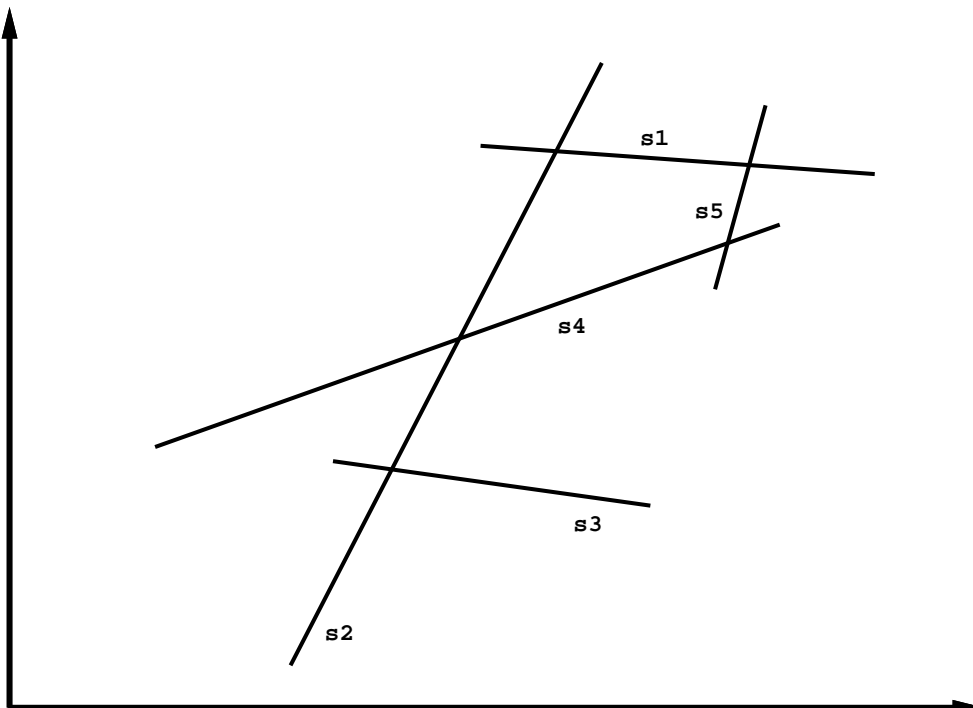


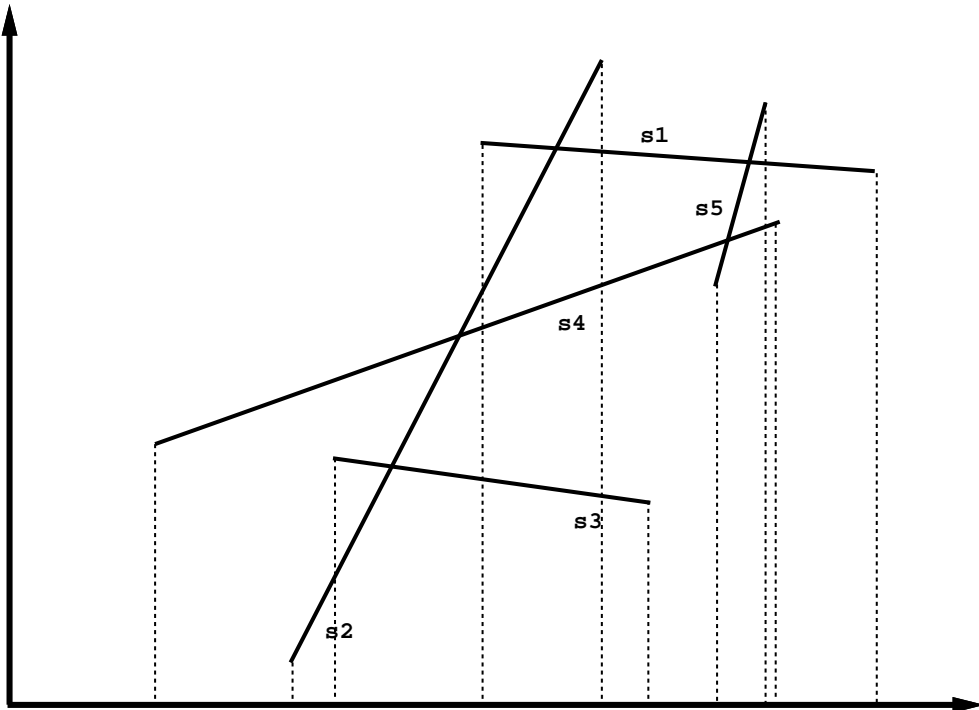
Intersection of Line Segments

- Given: n line segments.
- Find: All intersections (report segments at each intersection point)



- Trivial method: $O(n^2)$
- Lower bound: $\Omega(k + n \log n)$
- Plane-sweep: $O((n + k) \log n)$ time and $O(n + k)$ space.
- Necessary condition for two line segment to intersect: They must be next to each other on some vertical line.
- Simplifying assumptions:
 - No vertical segments.
 - No segments overlap.
 - No three segments meet at a common point.

Plane-Sweep Algorithm - Events and Status



- Events at x -coordinates of:
 - Left endpoints of segments
 - Right endpoints of segments
 - Intersections of line segments
- Status at given x -coordinate:
 - Segments intersected by a vertical line through x from top to bottom.

Plane-Sweep Algorithm

- Arriving at a left endpoint p_l of a segment s :
 - Insert s into the correct place in data structure.
 - Check if s intersects the segment above it. If so, add the intersection to the event list.
 - Check if s intersects the segment below it. If so, add the intersection to the event list.
- Arriving at a right endpoint p_r of a segment s :
 - Check if the segments above and below s intersect to the right of p_r . If so, and the intersection was not reported earlier, add it to the event list.
 - Remove s from the status data structure.
- Arriving at an intersection point p between two line-segments s_1 and s_2 .
 - Swap s_1 and s_2 in the data structure.
 - Check if s_2 and the segment above it intersect to the right of p . If so, and the intersection was not reported earlier, add it to the event list.
 - Check if s_1 and the segment below it intersect to the right of p . If so, and the intersection was not reported earlier, add it to the event list.

Plane-Sweep Algorithm - Data Structures

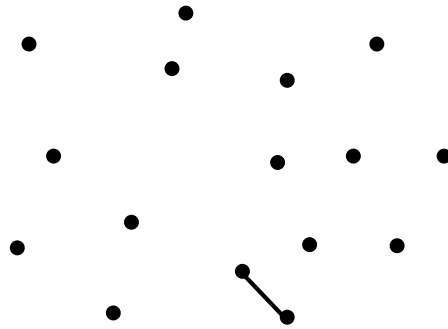
- Events data structure must permit the following operations:
 - Get the smallest (lexicographically) element and delete it.
 - Insert a new element in an appropriate position.
 - Check for membership.
- Status data structure must permit the following operations:
 - Insert an element in appropriate position.
 - Delete an element.
 - Swap to consecutive elements.
 - Return the element immediately before a given element.
 - Return the element immediately after a given element.
- Use balanced binary search trees both for events and for status.
- All operations can be carried out in $O(\log n)$ time.

Plane-Sweep Algorithm - Complexity

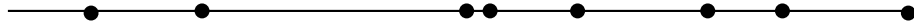
- Preprocessing, i.e., constructing event data structure: $O(n \log n)$ time and $O(n)$ space.
- Plane sweep: $O((n + k) \log n)$ time and $O(n + k)$ space.
- Plane sweep: $O((n + k) \log n)$ time.
 - Consider segments as a plane graph. It has endpoints and intersections as vertices.
 - The total number of vertices is $2n + k$.
 - The total number of edges is $O(n + k)$
 - Each edge is reported at most twice.
- Improvement to $O(n)$ space.
 - Delete event points corresponding to non-adjacent intersections.

Closest Pair Problem

- **Given:** n points in the plane.
- **Find:** closest pair.



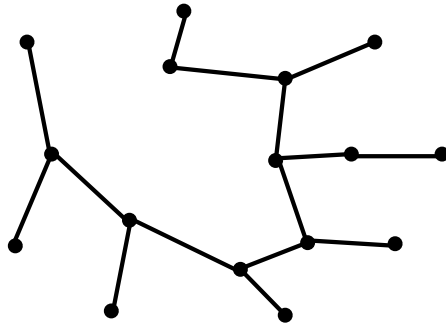
- Trivial algorithm $O(n^2)$
- Can it be improved?
- Yes, in 1 dimension.



- Sort. Closest pair is next to each other.
- Sorting $O(n \log n)$. Scanning $O(n)$ time.

Minimum Spanning Tree Problem

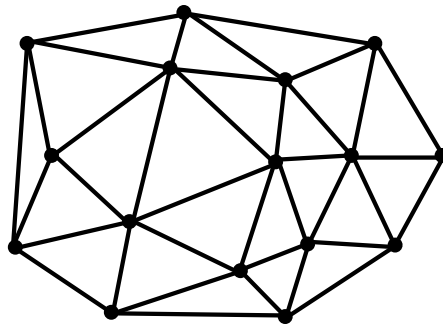
- **Given:** n points in the plane.
- **Find:** minimum spanning tree



- Can be solved by well-known methods for minimum spanning trees in weighted graphs (in the complete graph K_n with distances as edge weights).
- Is it possible to prune K_n ?
- Only pairs relatively close to each other need to be considered.

Delaunay Triangulation

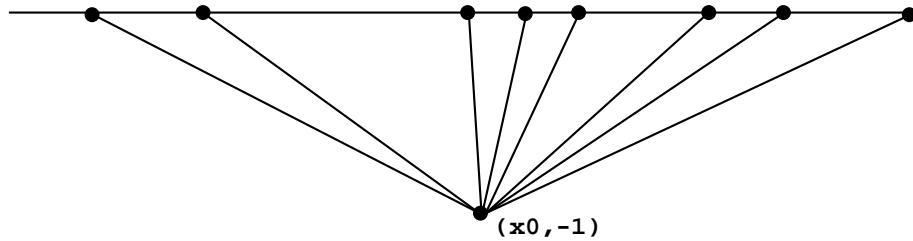
- **Given:** n points in the plane.
- **Add:** non-crossing edges so that all faces are triangular. The exterior face is the convex hull of the point set.



- Every triangulation has $3n - 6$ edges.
- There are many different triangulations:
 - minimum weight triangulation,
 - maximized smallest angle.

Lower Bounds

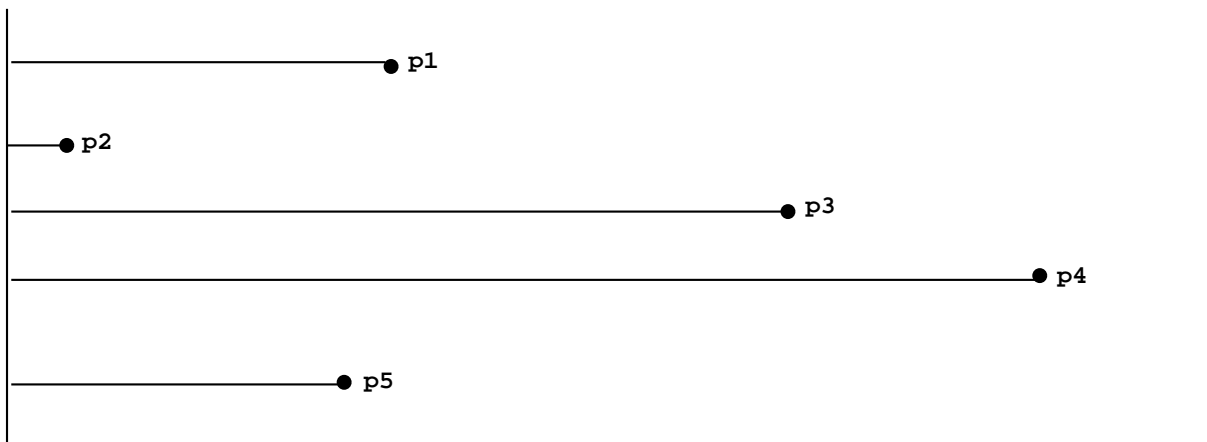
- Minimum spanning tree problem is $\Omega(n \log n)$; it is a generalization of sorting of n numbers.
 - Transformation $x \rightarrow (x, 0)$. Minimum spanning tree for this point-set is a path (defining the ordering).
- Triangulation is a generalization of sorting.



- edges incident with $(x_0, -1)$ give the ordering.

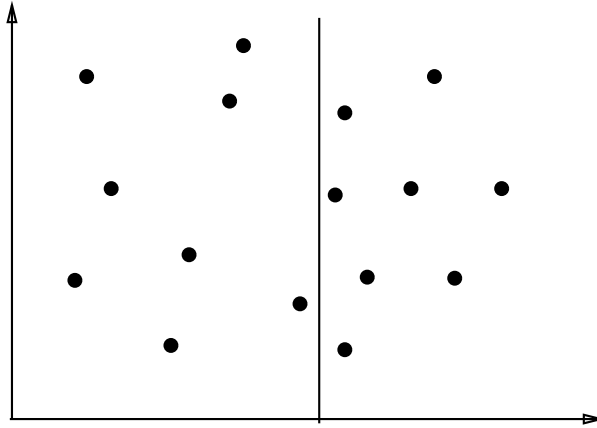
Closest Pair Problem

- Sorting provides an optimal $\Theta(n \log n)$ algorithm in 1-dimensional space.
- Can this be generalized to higher dimensions?
- Project onto one of the axes and then sort.



- Does not work. p_1 and p_5 are nearest neighbors but their projections are farthest away on the y -axis.

Closest Pair Problem - Divide and Conquer



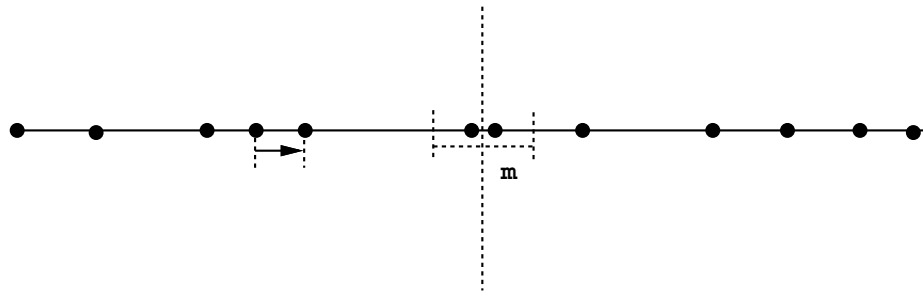
- Nearest neighbors in S_1 .
- Nearest neighbors in S_2 .
- Nearest neighbors, one in S_1 other in S_2 .
- Time complexity:

$$T(n) = 2T(n/2) + O(n^2/4)$$

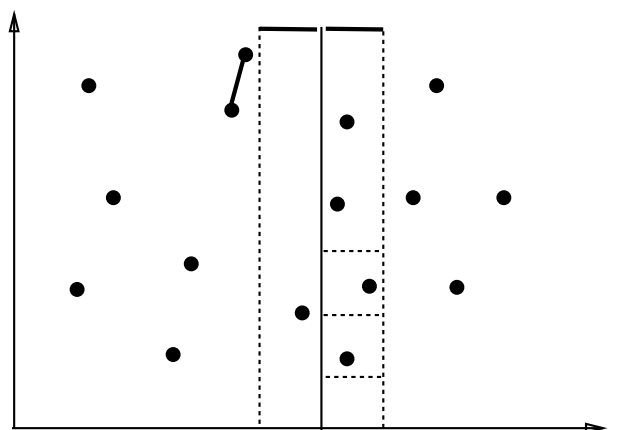
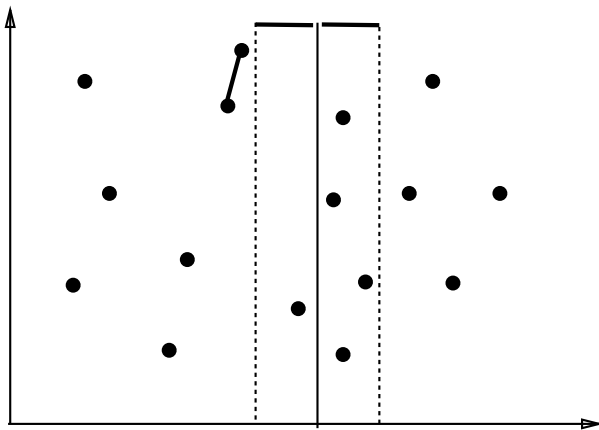
is $O(n^2)$

Closest Pair Problem - Divide and Conquer

- Is it necessary to check all $n^2/4$ pairs with one point in S_1 and the other point in S_2 ?
- In 1-dimensional space.

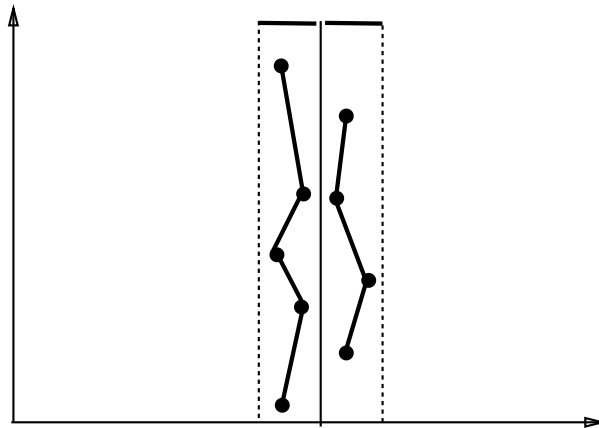


- Let $\sigma = \min\{|p_i p_j|, |q_k q_l|\}$
- Only points at distance σ need to be checked.
- There is at most one point in S_1 at distance σ from m . Similarly for S_2 .
- In 2-dimensional space.



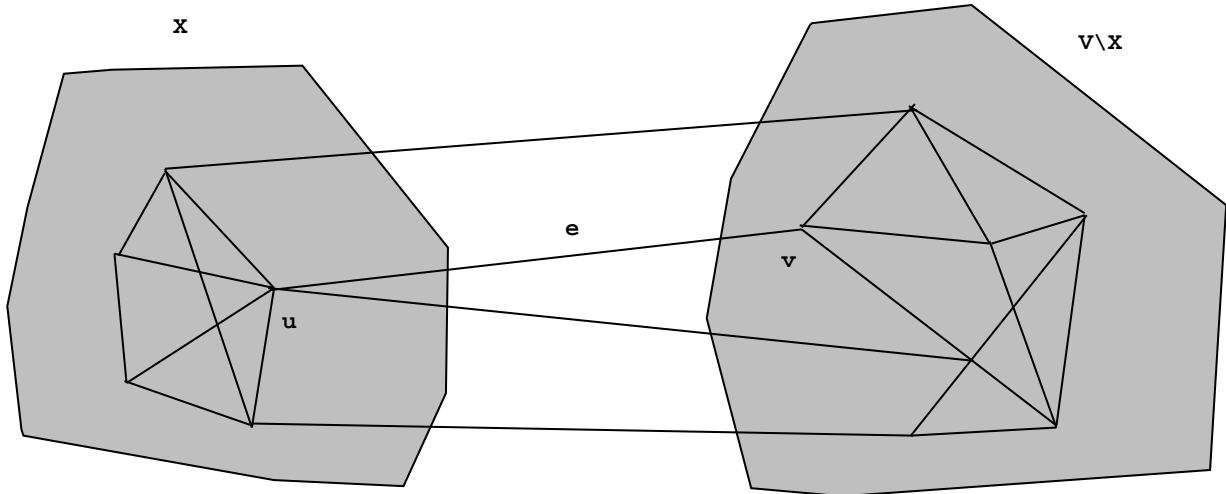
Closest Pair Problem - Divide and Conquer

- Preprocessing: Sort S by y -coordinates.
- Divide S into two equal size subsets S_1 and S_2 by a vertical median.
- Solve (recursively) for S_1 and S_2 . Let $\delta = \min\{\delta_1, \delta_2\}$ where δ_i is the smallest distance in S_i , $i = 1, 2$.
- Determine the upward chain P_i through points of S_i at distance δ from the median. Can be done in $O(n)$ time.

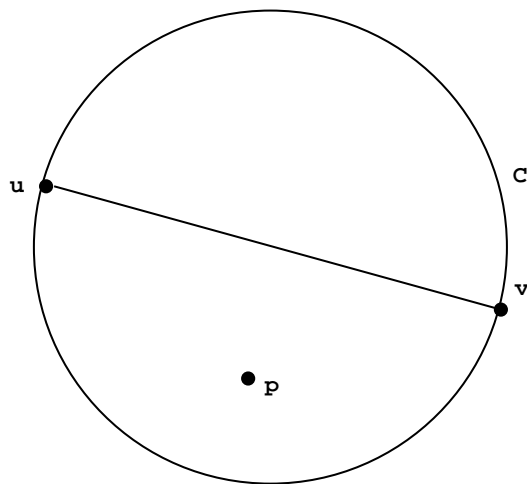


- In total $\Theta(n \log n)$.
- This method cannot be generalized to solve other problems.

Minimum Spanning Tree



- e is shortest crossing edge,
- e is not in DT



- p is either in X or in $V - X$
- $|up| < |uv|$ and $|vp| < |uv|$
- uv cannot be a crossing edge.