

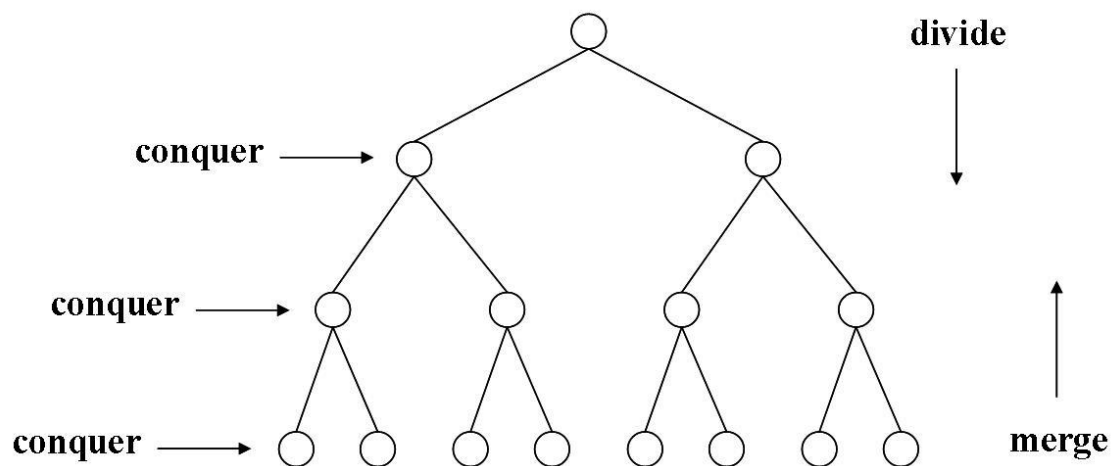
Divide-and-Conquer

Three main steps :

1. divide;

2. conquer;

3. merge.



Let I denote the (sub)problem instance and S be its solution. The divide-and-conquer strategy can be described as follows.

Procedure divide-and-conquer (I, S);

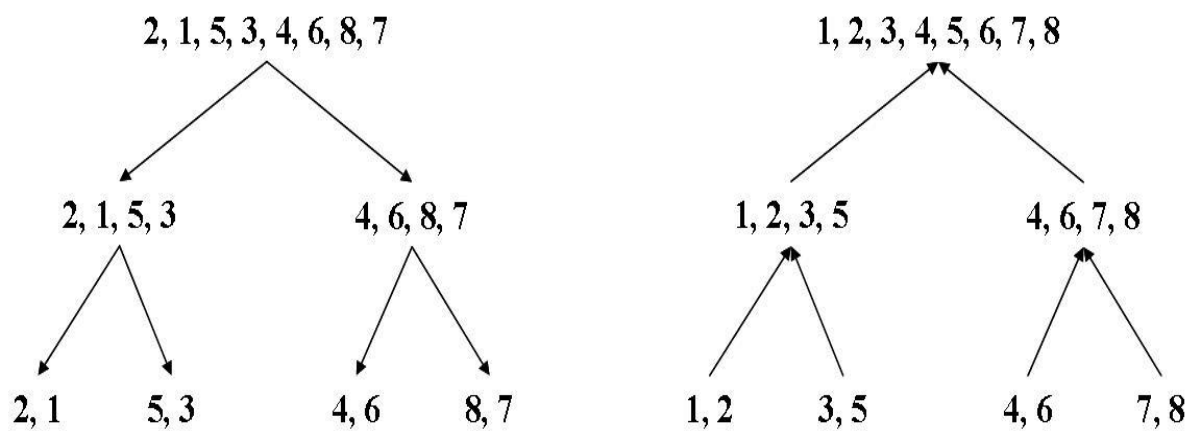
$O(1)$ if size(I) is small enough, then $S \leftarrow \text{solution}(I)$
else begin
 T_D split I into I_1, I_2, \dots, I_k ;
for $i \leftarrow 1$ to k
 T_i divide-and-conquer(I_i, S_i);
 T_M $S \leftarrow \text{merge}(S_1, S_2, \dots, S_k)$
end.

$$T = T_D + \sum_{i=1}^k T_i + T_M$$

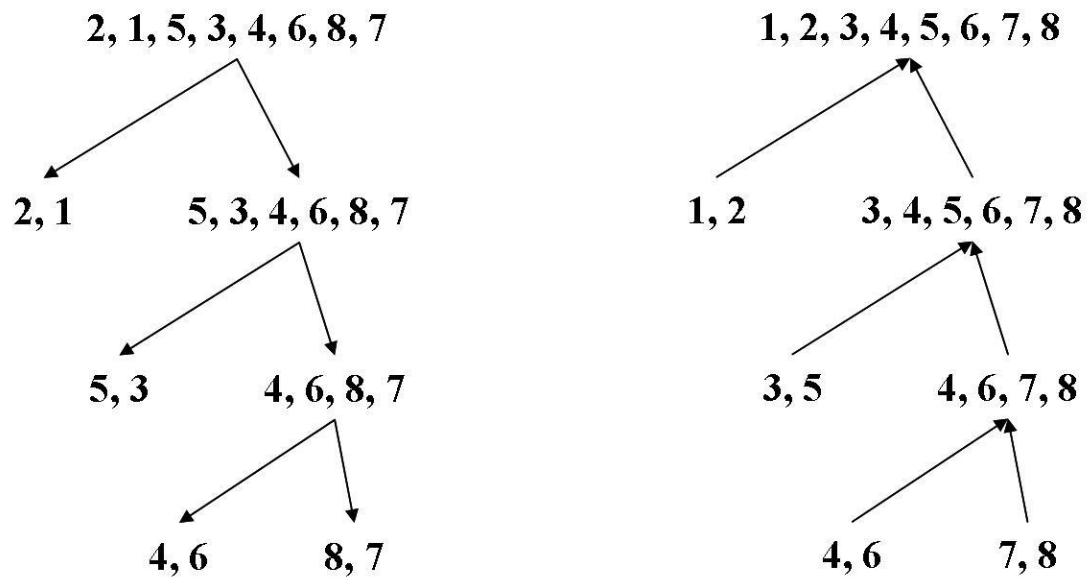
If size(I) = n , $k = 2$ and size(I_1) = size(I_2), then
 $T(n) = 2T(n/2) + T_D + T_M$.

Ex. Merge sort.

(Assume that $m + n$ time steps are required to merge two sorted lists of lengths m and n , respectively.)



16 time steps are required for merging.



18 time steps are required for merging.

⇒ a problem instance is always split into equal-size subproblem instances.

Ex. Compute $a_1 + a_2 + \dots + a_n$.

$$\begin{aligned} T(n) &= 2T(n/2) + O(1) \\ &= O(n) \end{aligned}$$

Ex. Merge sort of a_1, a_2, \dots, a_n .

$$\begin{aligned} T(n) &= 2T(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

Ex. Matrix multiplication

$$C_{n \times n} = A_{n \times n} \times B_{n \times n}$$

- divide

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \times \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$A_{ij}, B_{ij}, C_{ij}, : n/2 \times n/2$$

- conquer

$$T_1 = A_{11}B_{11}, \quad T_2 = A_{12}B_{21}, \quad T_3 = A_{11}B_{12}, \quad T_4 = A_{12}B_{22}$$

$$T_5 = A_{21}B_{11}, \quad T_6 = A_{22}B_{21}, \quad T_7 = A_{21}B_{12}, \quad T_8 = A_{22}B_{22}$$

- merge

$$C_{11} = T_1 + T_2, \quad C_{12} = T_3 + T_4, \quad C_{21} = T_5 + T_6,$$

$$C_{22} = T_7 + T_8$$

$$T(n) = 8T(n/2) + O(n^2)$$

$$= O(n^3)$$

Strassen's Method

- conquer

$$P = (A_{11} + A_{22}) \times (B_{11} + B_{22}) \quad Q = (A_{21} + A_{22}) \times B_{11}$$

$$R = A_{11} \times (B_{12} - B_{22}) \quad S = A_{22} \times (B_{21} - B_{11})$$

$$T = (A_{11} + A_{12}) \times B_{22} \quad U = (A_{21} - A_{11}) \times (B_{11} + B_{12})$$

$$V = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

- merge

$$C_{11} = P + S - T + V$$

$$C_{12} = R + T$$

$$C_{21} = Q + S$$

$$C_{22} = P + R - Q + U$$

$$T(n) = 7T(n/2) + O(n^2)$$

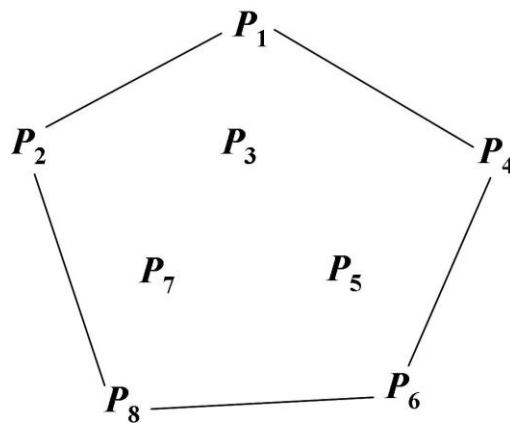
$$= O(n^{\log_2 7})$$

$$\approx O(n^{2.81})$$

Ex. The Convex Hull Problem.

Given a set S of n points in the plane, find its convex hull.

The convex hull of S is the smallest convex set that contains all the points of S .



The problem is required to report the vertices of the convex hull in either a clockwise sequence or a counterclockwise sequence, e.g., p_2, p_1, p_4, p_6, p_8 or p_6, p_4, p_1, p_2, p_8 .

Graham's Scan

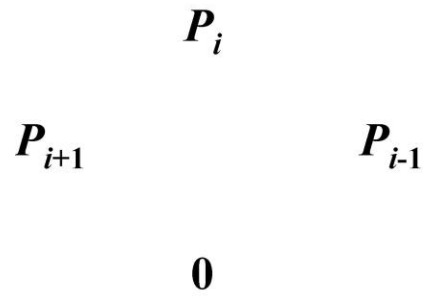
Step 1. Find an interior point O arbitrarily. $O(1)$

**Step 2. Sort all the points in angular order about O .
 $O(n \log n)$**

**Step 3. Select a point (assume p_1) that is on the convex
hull. $O(n)$**

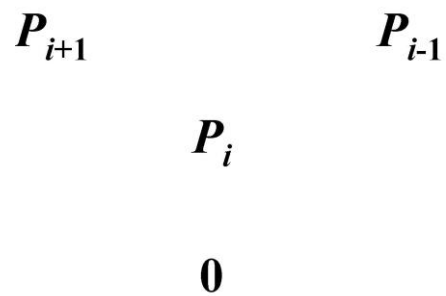
**Step 4. Start from p_1 and scan the sorted list of points.
Three contiguous points are examined at a
time. $O(n)$**

Case 1.



Do nothing, and examine (p_{i+2}, p_{i+1}, p_i) next.

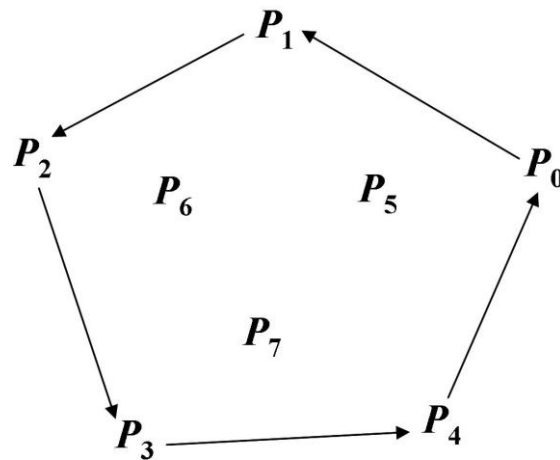
Case 2.



p_i is deleted, and examine $(p_{i+1}, p_{i-1}, p_{i-2})$ next.

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

Jarvis' March



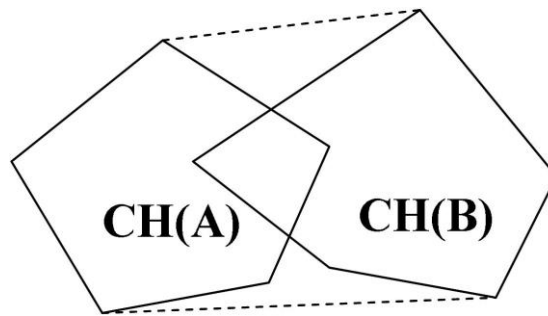
Step 1. Select a point (assume p_0) that is on the convex hull. $O(n)$

Step 2. Start from p_0 and find all the other convex hull vertices sequentially in a clockwise or counterclockwise order. $O(hn)$

$T(n) = O(hn)$, where h is the number of convex hull vertices.

Divide-and-Conquer Approach

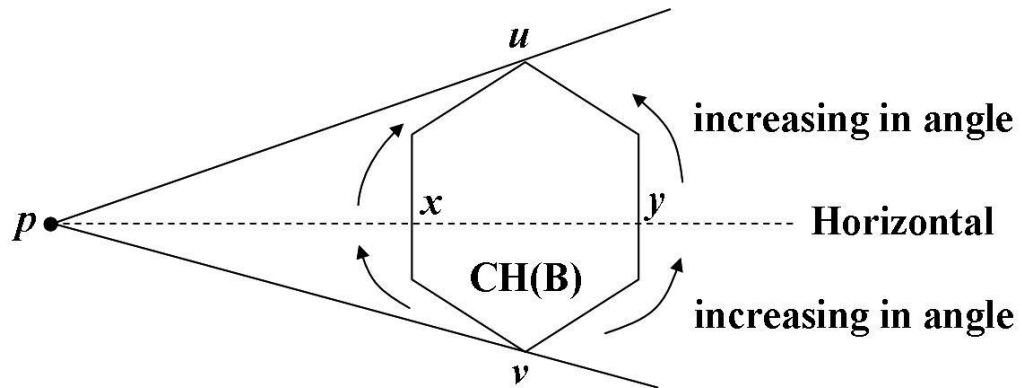
- **divide** : Divide the point set into two subsets A, B of approximately equal size. $O(1)$
- **conquer** : Find the convex hulls $CH(A), CH(B)$ of A and B , recursively. $2T(n/2)$
- **merge** : Find the convex hull of the original point set from $CH(A)$ and $CH(B)$. $O(n)$



$$T(n) = 2T(n/2) + O(n) = O(n \log n)$$

- (1) Find an interior point (assume p) of $\text{CH}(A)$ arbitrarily. $O(1)$**
- (2) Determine whether p is interior to $\text{CH}(B)$ or not. If not, go to (4). $O(n)$**
- (3) (p is interior to $\text{CH}(B)$) Since the vertices of $\text{CH}(A)$ and the vertices of $\text{CH}(B)$ are arranged in a sorted angular order about p , we may obtain a sorted list of their union by merging their respective sorted lists. Go to (5). $O(n)$**
- (4) (p is not interior to $\text{CH}(B)$) Find two vertices u, v of $\text{CH}(B)$ so that the convex wedge defined by u, p and v contains $\text{CH}(B)$. $O(n)$**

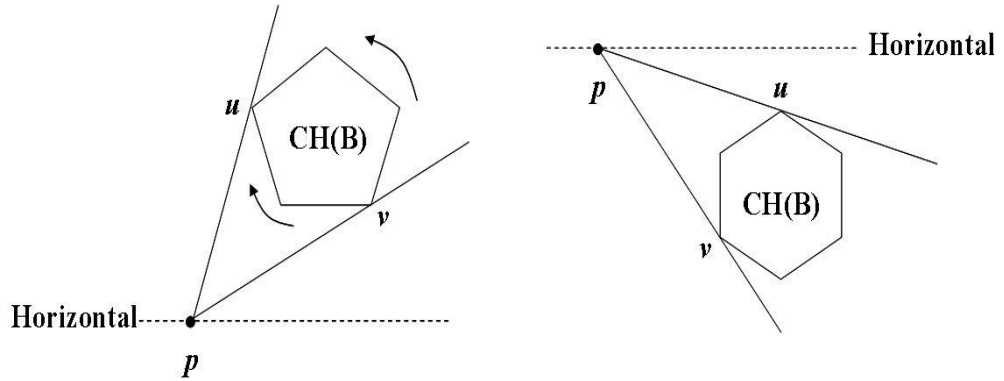
Case 1.



List 1 : list of vertices arranged in their occurring sequence from x to u and then from v to x .
(Assume that the degrees are measured between 0 and 2π .)

List 2 : list of vertices arranged in their occurring sequence from y to u and then from v to y .

Case 2.



List 1 : list of vertices arranged in their clockwise occurring sequence from v to u .

List 2 : list of vertices arranged in their counterclockwise occurring sequence from v to u .

Since List 1 and List 2 are increasing in polar angle about p , we may merge them and vertices of $CH(A)$ into a sorted list.

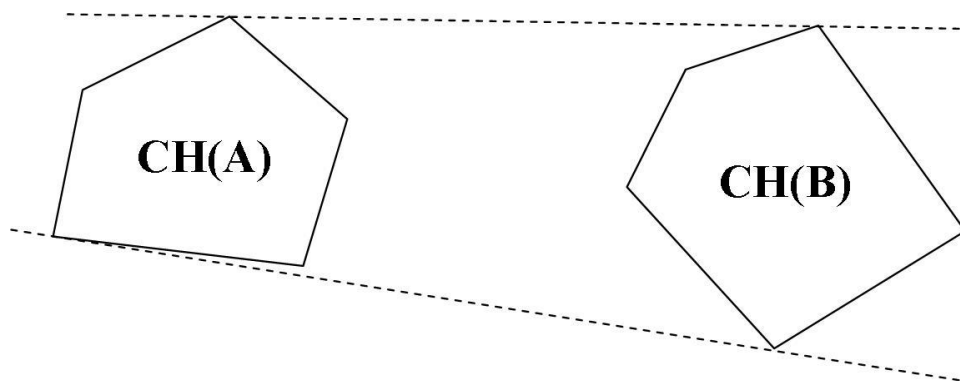
(5) Perform Step 3 and Step 4 of Graham's Scan to obtain $\text{CH}(A \cup B)$. $O(n)$

(Graham's Scan :

Step 3. Select a point (assume p_1) that is on the convex hull.

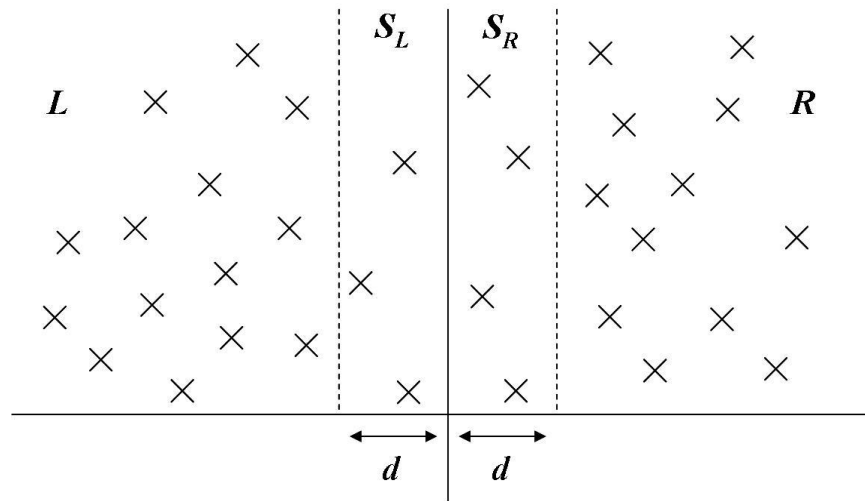
Step 4. Start from p_1 and scan the sorted list of points. Three contiguous points are examined at a time.)

N.B. There is another divide-and-conquer approach to the convex hull problem, which divides the point set into two approximately equal-size subsets A and B by means of a vertical line. $CH(A \cup B)$ is obtained by determining two common supporting lines of $CH(A)$ and $CH(B)$. Refer to “Convex hulls of finite sets of points in two and three dimensions”, *Comm. ACM*, vol. 20, no. 2, 1977, 87-93. (by Preparata and Hong)

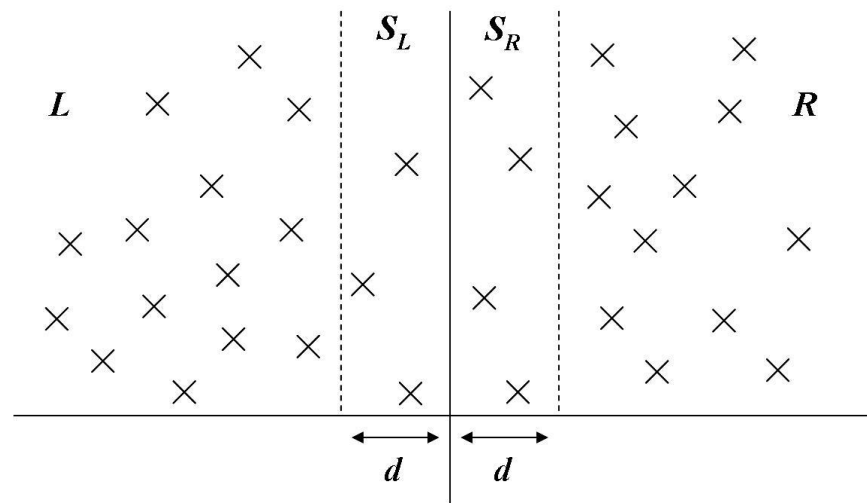


Ex. The 2-dimensional closest pair problem.

Given a set of n points in the plane, find the pair of points that are closest.

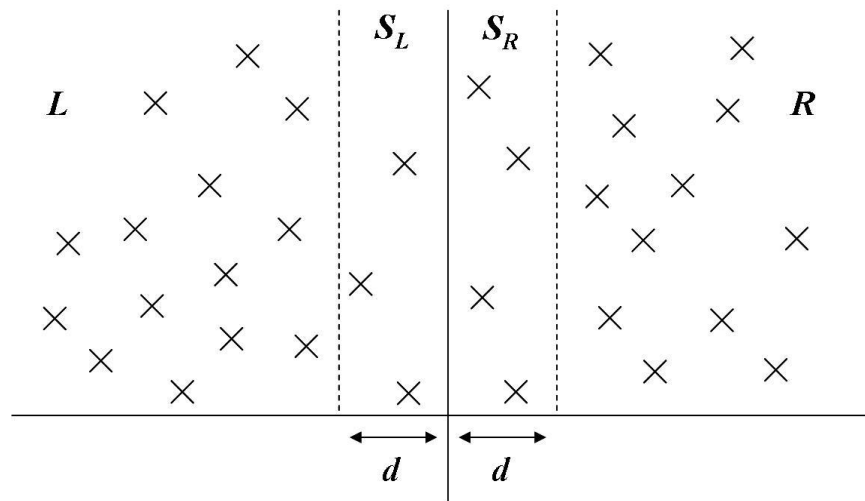


- **divide** : Find a line perpendicular to the x -axis which divides the point set into two equal-size subsets L and R (i.e., find the median of all the x -coordinates). $O(n)$

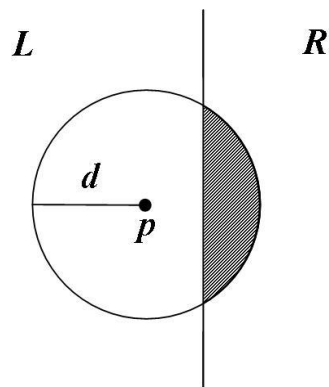


- **conquer** : Find the closest pairs in L and R , respectively. Let d_L and d_R be their respective distances. $2T(n/2)$

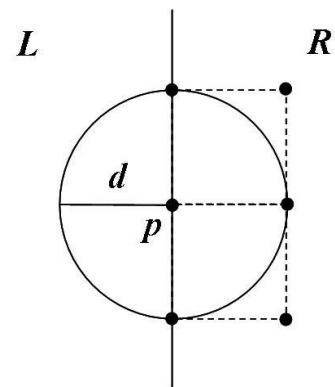
- merge : Let $d = \min\{d_L, d_R\}$. If the closest pair of $L \cup R$ consists of a point in L and a point in R , then they must lie within a slab of width $2d$ centered at the dividing line. Thus, we only have to determine for each point p in L whether there exist points in R that are within d from p . $O(n)$



- (1) Presort all the given points according to their y -coordinates (before the divide-and-conquer procedure is initiated). $O(n \log n)$
- (2) Examine the two sorted lists of points that are located in S_L and S_R , respectively. For each point p in L , we only need to examine at most five points in R . $O(n)$



Only the shaded area has to be examined.



At most six points fit in a $d \times 2d$ rectangle.

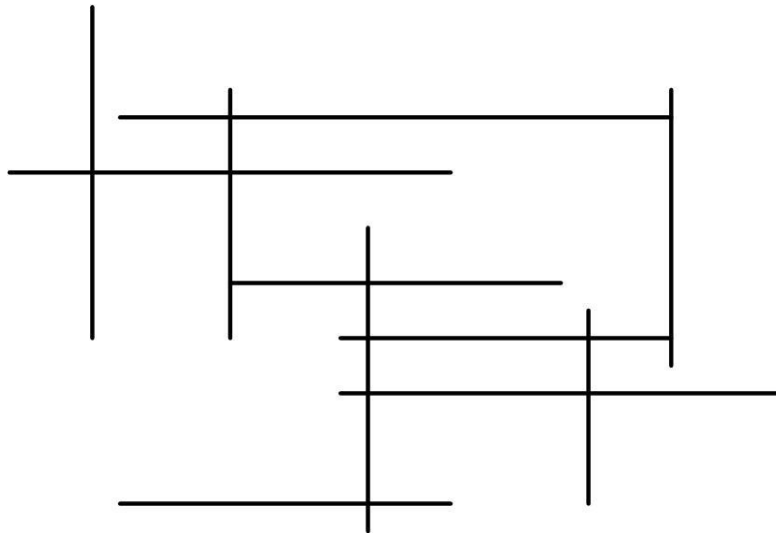
presorting time : $O(n \log n)$

$$\begin{aligned} T(n) &= 2T(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

$$\begin{aligned} \text{total time complexity} &= \text{presorting time} + T(n) \\ &= O(n \log n) \end{aligned}$$

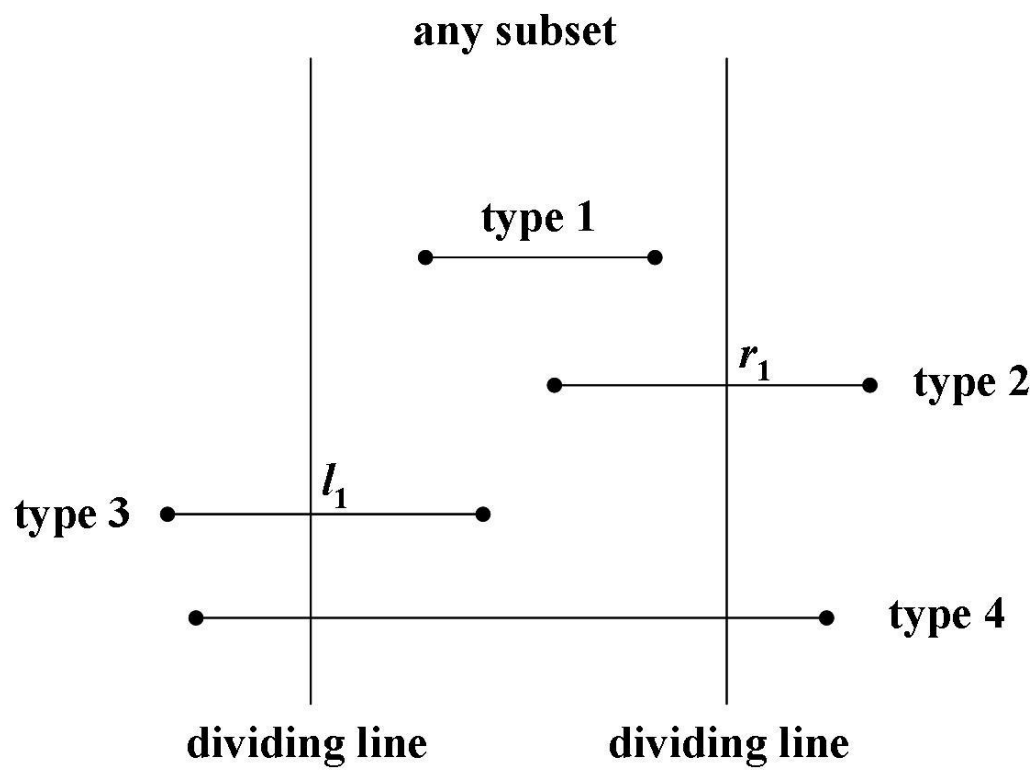
Ex. The Iso-Oriented Line Segment Intersection Problem.

Given a set of m horizontal and vertical line segments L_1, L_2, \dots, L_m , find all pairs of intersecting line segments.



Let objects refer to either vertical line segments or endpoints of horizontal line segments. Assume that there are n objects initially.

- **Presort all the endpoints of vertical line segments according to their y -coordinates. $O(n \log n)$**
- **divide : Divide the set of objects into two approximately equal-size subsets L and R by a vertical line. Also split the sorted list above in accordance with L and R . $O(n)$**

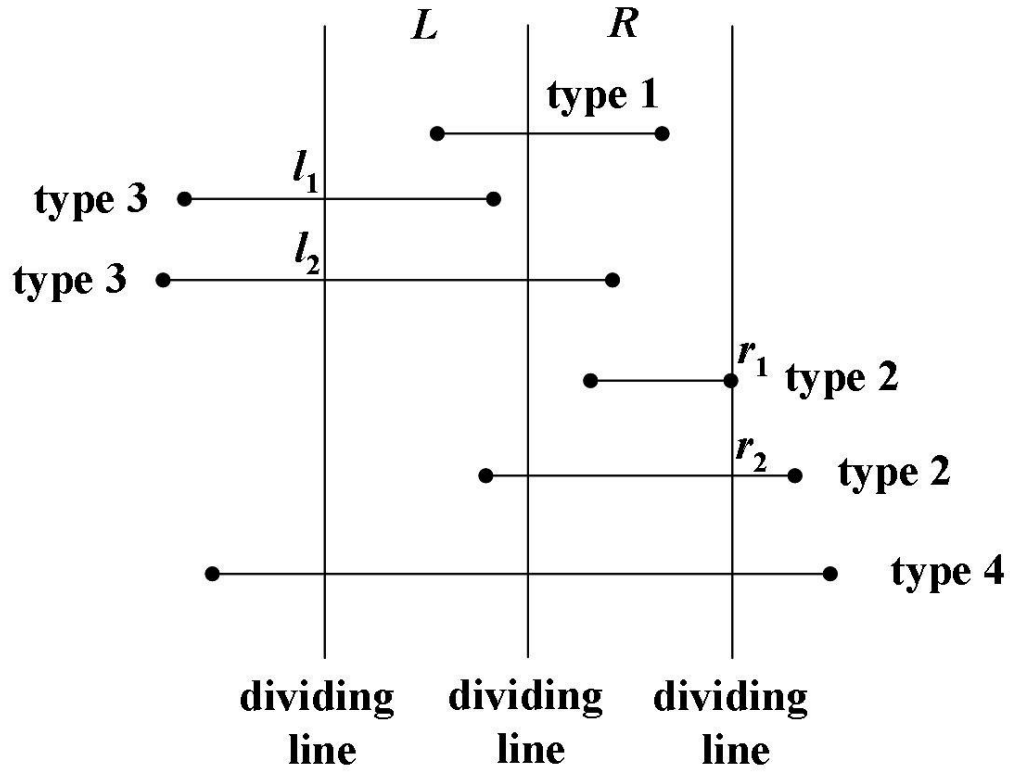


- conquer : $2T(n/2)$

For each subset (the existence of line segments of type 4 is unknown),

- (1) report all intersecting pairs in which only type 1, type 2 and type 3 horizontal line segments are included;
- (2) record right boundary points r_1, r_2, \dots (denoted by list \tilde{R}) in sorted y -coordinate sequence;
- (3) record left boundary points l_1, l_2, \dots (denoted by list \tilde{L}) in sorted y -coordinate sequence.

- merge : $O(n) + s$



\tilde{R}_L (\tilde{R}_R) : list \tilde{R} of the left (right) subset

\tilde{L}_L (\tilde{L}_R) : list \tilde{L} of the left (right) subset

$\tilde{L}\tilde{R}$: list of y-coordinates of type 1 line segments
(across the middle dividing line)

V_L (V_R) : set of vertical line segments in the left
(right) subset

$$\tilde{L}\tilde{R} \leftarrow \tilde{R}_L \cap \tilde{L}_R$$

$$\tilde{R} \leftarrow \tilde{R}_R + (\tilde{R}_L - \tilde{L}\tilde{R})$$

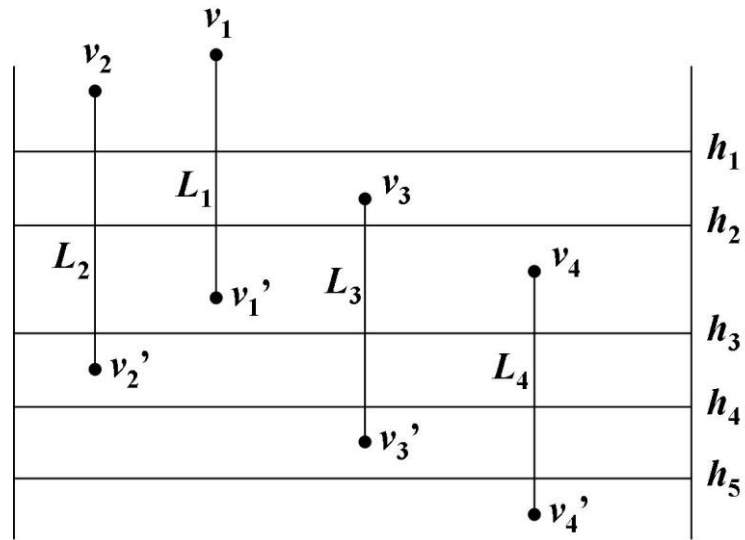
$$\tilde{L} \leftarrow \tilde{L}_L + (\tilde{L}_R - \tilde{L}\tilde{R})$$

Report all intersecting pairs induced by

- ♣ V_L and the horizontal line segments
passing $\tilde{L}_R - \tilde{L}\tilde{R}$;
- ♣ V_R and the horizontal line segments
passing $\tilde{R}_L - \tilde{L}\tilde{R}$

Since \tilde{R}_L , \tilde{R}_R , \tilde{L}_L and \tilde{L}_R are sorted in y-coordinate, we may obtain $\tilde{L}\tilde{R}$, \tilde{L} and \tilde{R} (sorted in y-coordinate) in $O(n)$ time by performing merge-like operations.

A method to report all intersecting pairs in $O(n + s)$ time, where s is the number of intersecting pairs reported :



Given two sorted lists :

$$H = (h_1, h_2, h_3, h_4, h_5)$$

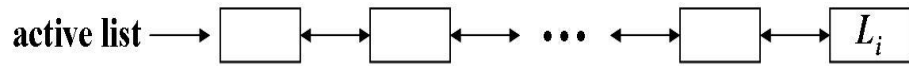
$$V = (v_1, v_2, v_3, v_4, v_1', v_2', v_3', v_4')$$

Step 1. Merge H and V into L .

$$L = (v_1, v_2, h_1, v_3, h_2, v_4, v_1', h_3, v_2', h_4, v_3', h_5, v_4')$$

Step 2. Scan L from the left to the right.

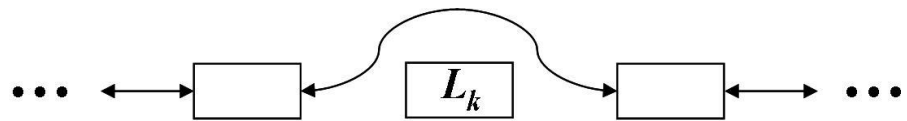
Case 1. Add L_i to the active list whenever a v_i is met.



Case 2. Report the active list whenever an h_j is met.

Each vertical line segment in the active list intersects with h_j .

Case 3. Delete L_k from the active list whenever a v_k' is met.



Another method to report all intersecting pairs in $O(n + s)$ time :

Given two sorted lists :

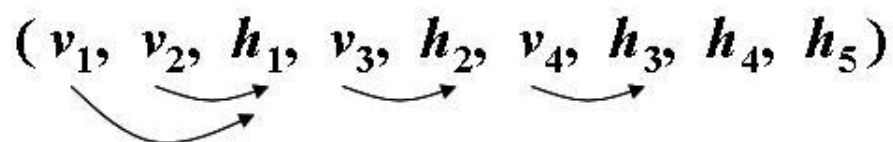
$$H = (h_1, h_2, h_3, h_4, h_5)$$

$$V = (v_1, v_2, v_3, v_4)$$

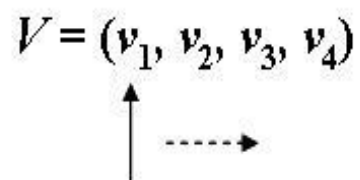
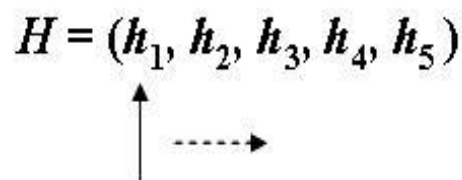
Step 1. Merge H and V into $L = (v_1, v_2, h_1, v_3, h_2, v_4, h_3, h_4, h_5)$.

Step 2. Scan L from the right to the left.

For each v_i , find the h_j that is succeeding and closest to it.



Step 3. Report all intersecting pairs by scanning H and V simultaneously.



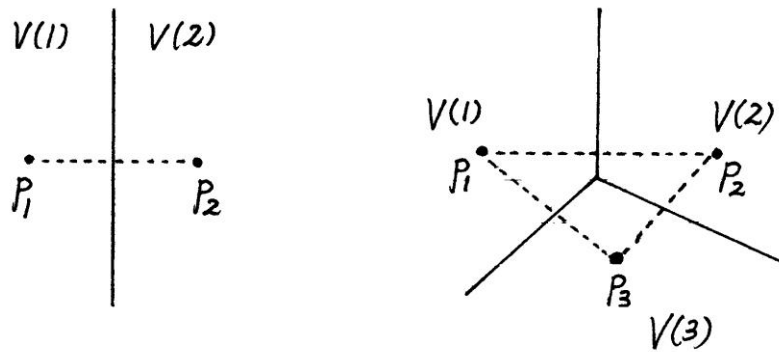
$$T(n) = 2T(n/2) + O(n) = O(n \log n)$$

$$\begin{aligned} \text{total time complexity} &= T(n) + O(n \log n) + O(N) \\ &= O(n \log n + N), \end{aligned}$$

where $T(n)$ is the time for D&C, $O(n \log n)$ is the time for presorting, and N is the total number of intersecting pairs.

Ex. Voronoi Diagram Construction.

Given n points p_1, p_2, \dots, p_n in the plane, we denote by $V(i)$ the locus of points that are closer to p_i than to any other point.

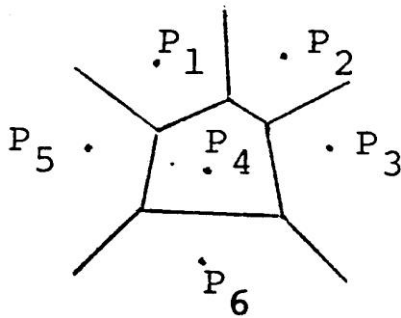


Let $H(p_i, p_j)$ denote the half-plane where all the points are closer to p_i than to p_j . Thus, $V(i)$ is a convex polygonal region having no more than $n - 1$ sides, defined by

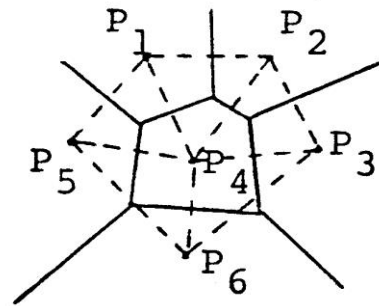
$$V(i) = \bigcap_{j \neq i} H(p_i, p_j).$$

$V(i)$ is called the *Voronoi polygon* associated with p_i . The partition of the plane by n Voronoi polygons is called the *Voronoi diagram*.

The straight-line dual of a Voronoi diagram on a set S of n points is a graph whose n vertices correspond to the points of S , in which there is an edge from p_i to p_j iff $V(i)$ and $V(j)$ share an edge.



**A Voronoi diagram
of six points**



A Delaunay Triangulation

The straight-line dual of a Voronoi diagram is a planar graph, called the *Delaunay graph*. If no four points of the original set are cocircular, the straight-line dual is a triangulation, called the *Delaunay triangulation* (proved by Delaunay at 1934).

Theorem. A Voronoi diagram on n points has at most $2n - 4$ vertices and $3n - 6$ edges.

Proof. (1) Each edge in the Delaunay graph corresponds to a unique Voronoi edge.

(2) A planar graph of n vertices has at most $3n - 6$ edges.

(3) Each Voronoi vertex has degree at least 3.

(1), (2) \Rightarrow there are at most $3n - 6$ Voronoi edges

\Rightarrow (with (3)) there are at most

$$2 \cdot (3n - 6) / 3 = 2n - 4$$

Voronoi vertices.

Theorem. A Voronoi polygon $V(i)$ is unbounded iff p_i lies on the boundary of the convex hull (of given points).

Proof. Refer to “Computational Geometry” by Shamos, Ph.D. dissertation, Yale Univ., 1978.

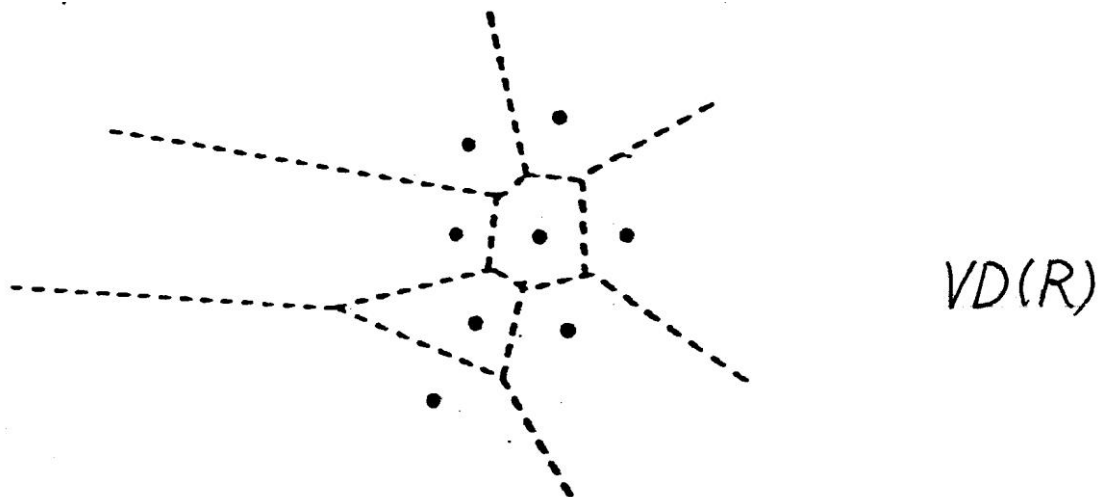
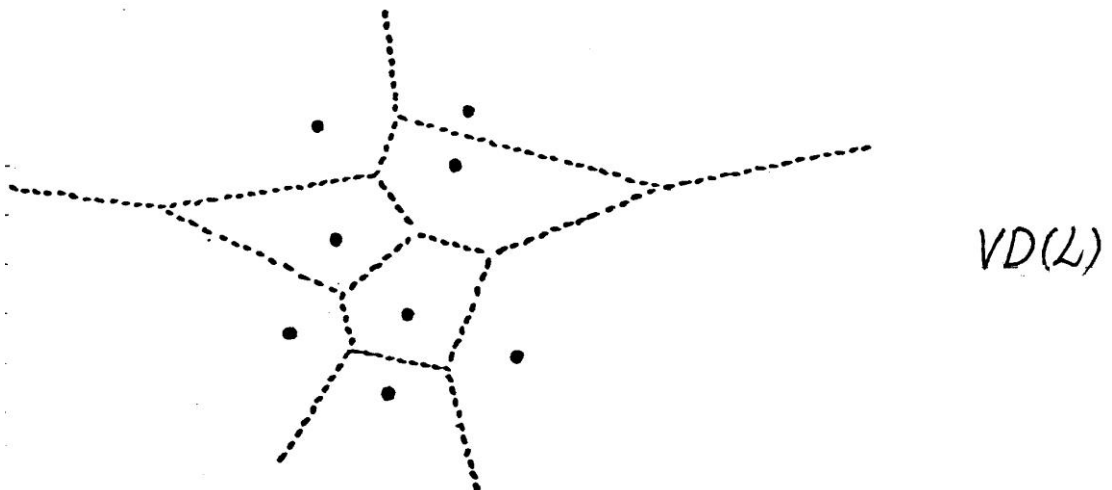
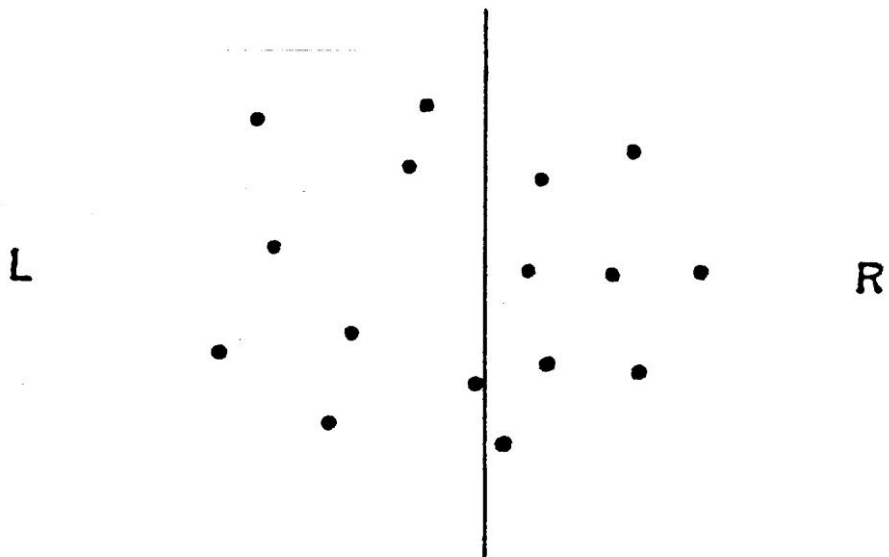
Constructing Voronoi diagram

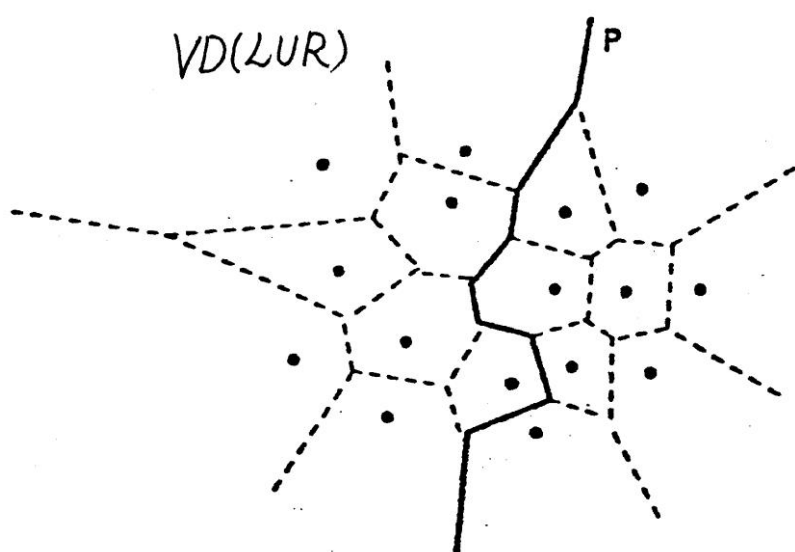
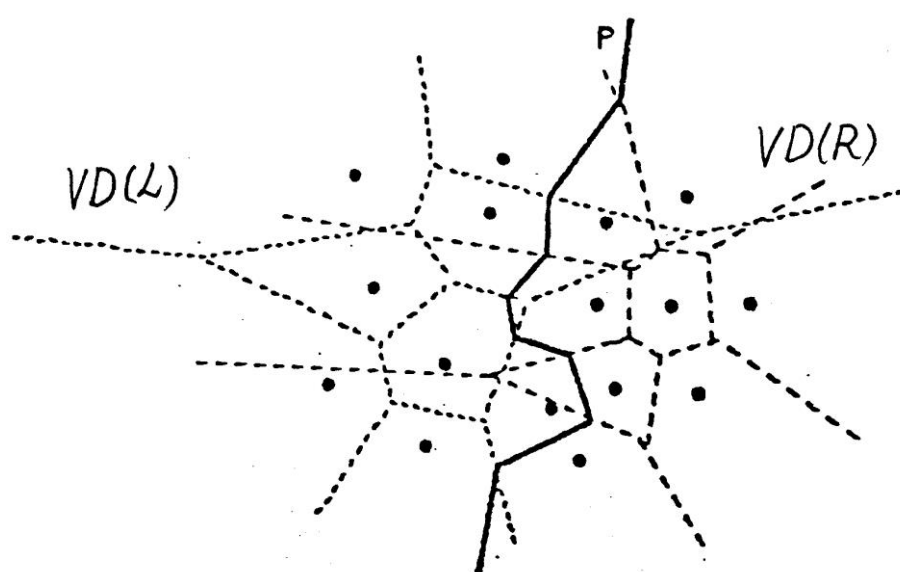
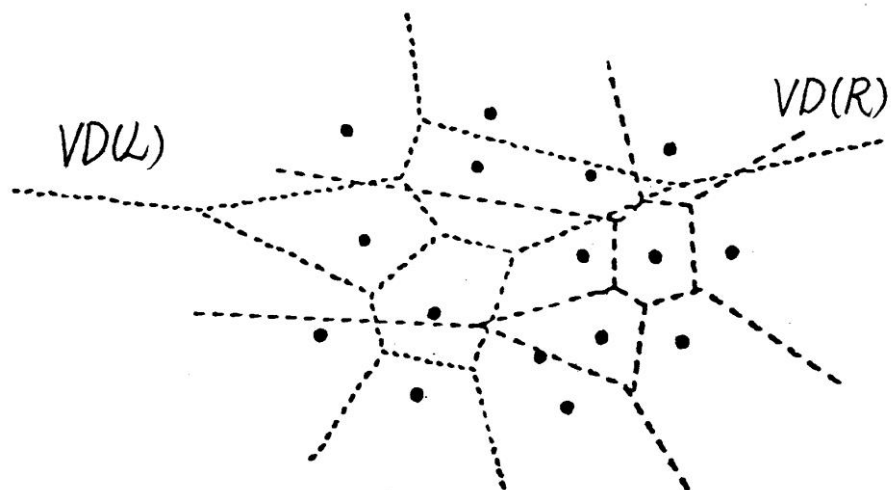
By “constructing” Voronoi diagram of a set of points, we mean obtaining all of the following data :

- (1) coordinates of Voronoi vertices;**
- (2) Voronoi edges (pairs of Voronoi vertices)
incident with each Voronoi vertex;**
- (3) the two points that determine each Voronoi
edge;**
- (4) the edges of each Voronoi polygon in a cyclic
order.**

- **divide :** Divide the given set of points into two subsets L and R of approximately equal size by a vertical line. $O(n)$
- **conquer :** Construct the Voronoi diagram, denoted by $VD(L)$, for L and the Voronoi diagram, denoted by $VD(R)$, for R . $2T(n/2)$
- **merge :** Find P , which is the locus simultaneously closest to a point in L and a point in R , and discard all line segments of $VD(R)$ ($VD(L)$) that lie to the left (right) of P . $O(n)$

dividing line





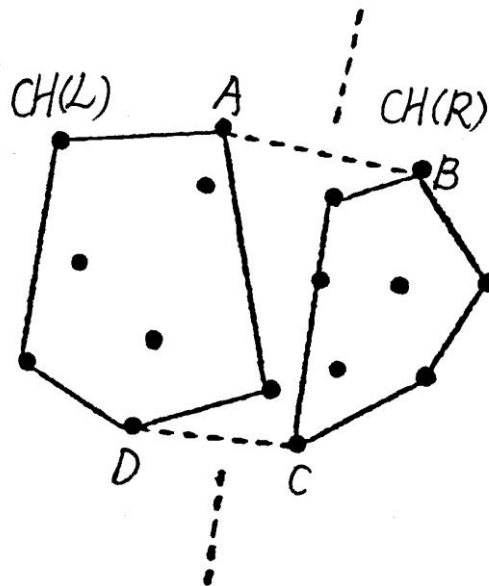
Theorem. P is monotonic and continuous in y -coordinate and consists of two infinite rays and some piecewise line segments.

Proof. Refer to “Computational Geometry” by Shamos, Ph.D. dissertation, Yale Univ., 1978.

Two steps to find P in $O(n)$ time :

Step 1. Find the two infinite rays. $O(n)$

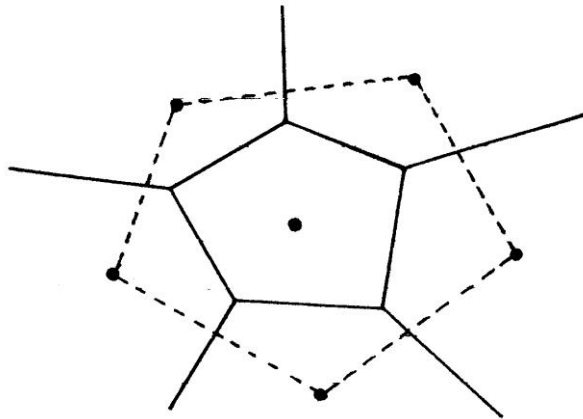
Since each ray is associated with two adjacent vertices in $CH(L \cup R)$, we find the two edges of $CH(L \cup R)$ that are not present in either $CH(L)$ or $CH(R)$.



The infinite rays of P are the perpendicular bisectors of the two segments joining $CH(L)$ and $CH(R)$.

1-1. Find $CH(L)$ and $CH(R)$. $O(n)$

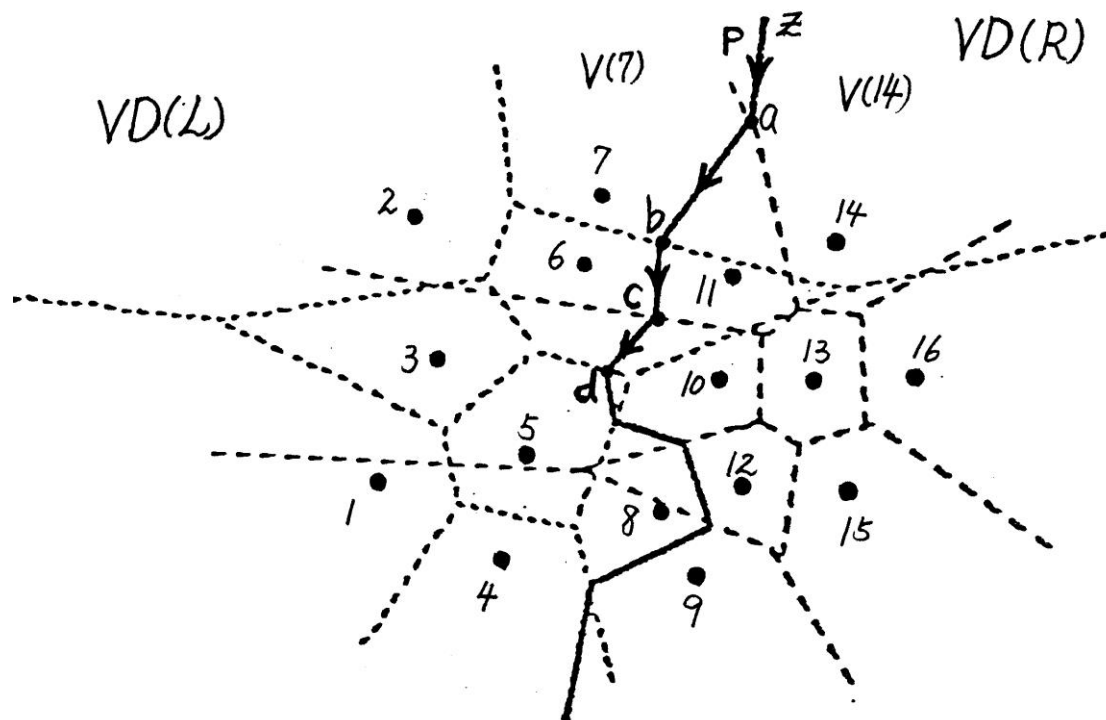
- (a) Find a ray arbitrarily.**
- (b) Find another ray belonging to the same Voronoi polygon.**
- (c) Perform (b) in clockwise or counter-clockwise orientation until the initial ray is met.**



1-2. Find A, B, C, D . $O(n)$.

The merge algorithm for the convex hull problem can be applied here.

Step 2. A zigzag walk to construct P . $O(n)$



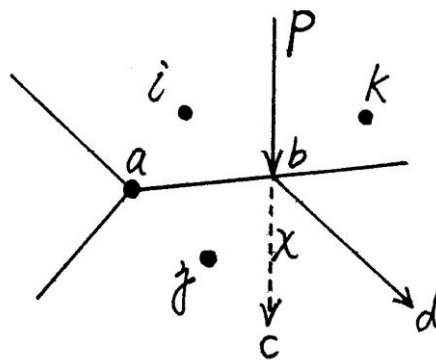
za has equal distance to 7 and 14 (za in $V(7) \cap V(14)$).

ab has equal distance to 7 and 11 (ab in $V(7) \cap V(11)$).

bc has equal distance to 6 and 11 (bc in $V(6) \cap V(11)$).

Two observations :

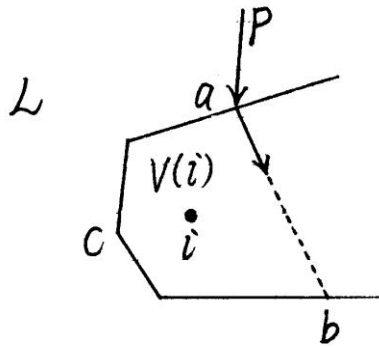
- 1. P always moves downward (since P is monotonic).**
- 2. P will bend toward the right (left) when it passes through an edge belonging to $VD(L)$ ($VD(R)$).**



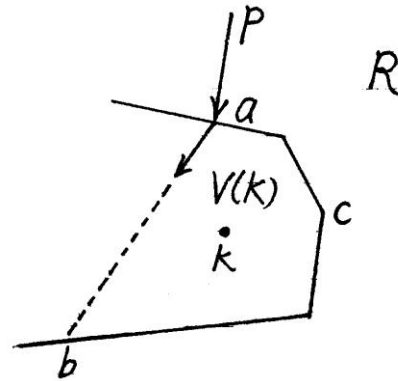
pb has equal distance to i and k .

bd has equal distance to j and k .

For each $x \in bc$, $jx < ix = xk$.



**P moves out of $V(i)$ at b
(if no edge belonging to
 $VD(R)$ is met before b).**



**P moves out of $V(k)$ at b
(if no edge belonging to
 $VD(L)$ is met before b).**

- (1) When P enters $V(i)$, we mark the edge containing b .
If P meets edges belonging to $VD(R)$ before leaving $V(i)$, we mark the edge of $V(i)$ where P will leave $V(i)$ if P continues in its changed direction. The newly marked edge can be found as we traverse the edges of $V(i)$ in clockwise orientation, starting from the last marked edge.**

**(2) When P enters $V(k)$, we mark the edge containing b .
If P meets edges belonging $VD(L)$ before leaving $V(k)$, we mark the edge of $V(k)$ where P will leave $V(k)$ if P continues in its changed direction. The newly marked edge can be found if we traverse the edges of $V(k)$ in counterclockwise orientation, starting from the last marked edge.**

Therefore, whenever P passes through an edge, we can find the edge that P will pass through next as follows :

determine the two currently marked edges that belong to $VD(L)$ and $VD(R)$, respectively, and choose the one that P will meet first.

Thus, $O(n)$ time is enough to construct P .

Applications with Voronoi diagram :

1. All nearest neighbors problem.

Given n points in the plane, find the nearest neighbor of each.

Theorem. The nearest neighbor of each p_i defines an edge of $V(i)$.

Proof. Refer to *Computational Geometry*, Springer-Verlag, by Preparata & Shamos.

Thus, given the Voronoi diagram, this problem can be solved in $O(n)$ time.

2. Nearest neighbor search problem.

Given n point in the plane, with preprocessing allowed, find the nearest neighbor of a query point.

- (1) Construct the Voronoi diagram of the given n points.**
- (2) Find the Voronoi polygon where the query point is located.**

Theorem. Point location in an n -vertex planar subdivision can be effected in $O(\log n)$ time using $O(n)$ storage, given $O(n \log n)$ preprocessing time.

Proof. Refer to *Computational Geometry*, Springer-Verlag, by Preparata & Shamos.

Thus, nearest-neighbor search for a query point can be performed in $O(\log n)$ time, using $O(n)$ storage, with $O(n \log n)$ preprocessing time.

3. Euclidean minimum spanning tree problem.

Given n point in the plane, construct a tree of minimum total length whose vertices are the given points.

If we construct a complete graph of n vertices, each corresponding to a point, then $O(n^2)$ time (by Prim's method) or $O(n^2 \log n)$ time (by Kruskal's method) is needed.

N.B. Prim's algorithm takes $O(|V|^2)$ time, and Kruskal's algorithm takes $O(|E| \cdot \log |E|)$ time.

Theorem. Every Euclidean minimum spanning tree is a subgraph of the Delaunay graph.

Proof. Refer to “Computational Geometry” by Shamos, Ph.D. Dissertation, Yale Univ., 1978.

Theorem. A minimum spanning tree of a planar graph can be found in $O(n)$ time.

Proof. Refer to “Finding minimum spanning trees” by Cheriton & Tarjan, *SIAM J. Computing*, vol. 5, no. 4, 1976, 724-742.

- 1. Construct the Voronoi diagram. $O(n \log n)$**
- 2. Construct the Delaunay graph. $O(n)$**
- 3. Find a minimum spanning tree on the Delaunay graph. $O(n)$**

A related work :

**H. Imai, M. Iri, and K. Murota, “Voronoi diagram
in the Laguerre geometry and its applications,”
SIAM J. Computing, vol. 14, no. 1, 1985, 93-105.**

Program Assignment 4 :

**Write an executable program to solve the
2-dimensional closest pair problem.**

Exercise 4 :

**Imai, Iri, and Murota, "Voronoi diagram in the
Laguerre geometry and its applications," *SIAM
Journal on Computing*, vol. 14, no. 1, 1985, pp.
93-105.**