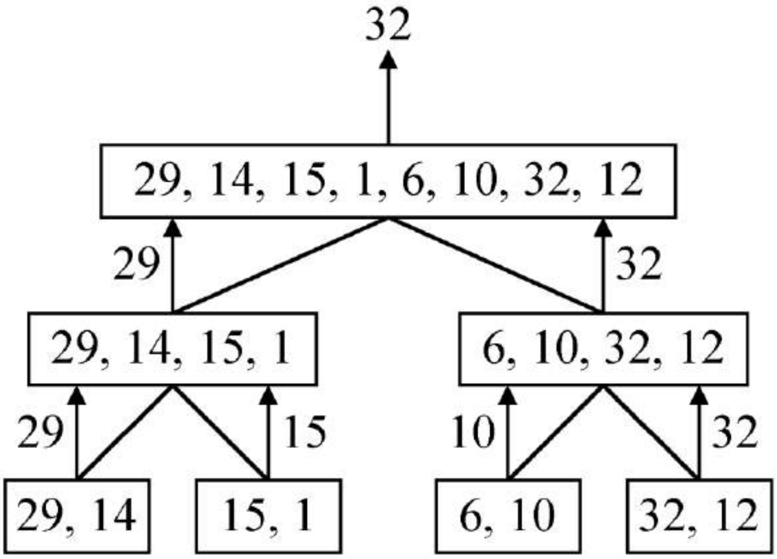
§The Divide-and-Conquer Strategy

e.g. find the maximum of a set S of n numbers



time complexity:

$$T(n)=\begin{cases} 2T(n/2)+1 & , n>2 \\ 1 & , n\leq 2 \end{cases}$$

assume $n = 2^k$

$$\begin{aligned} T(n) &= 2T(n/2)+1 \\ &= 2(2T(n/4)+1)+1 \\ &= 4T(n/4)+2+1 \\ &\quad \vdots \\ &=2^{k-1}T(2)+2^{k-2}+\cdots+4+2+1 \\ &=2^{k-1}+2^{k-2}+\cdots+4+2+1 \\ &=2^k-1 = n-1 \end{aligned}$$

A general divide-and-conquer algorithm:

Step 1: If the problem size is small, solve this problem directly; otherwise, split the original problem into 2 sub-problems with equal sizes.

Step 2: Recursively solve these 2 sub-problems by applying this algorithm.

Step 3: Merge the solutions of the 2 sub-problems into a solution of the original problem.

time complexity:

$$T(n)=\begin{cases} 2T(n/2)+S(n)+M(n) & , n \geq c \\ b & , n < c \end{cases}$$

where S(n): time for splitting

M(n): time for merging

b: a constant

c: a constant.

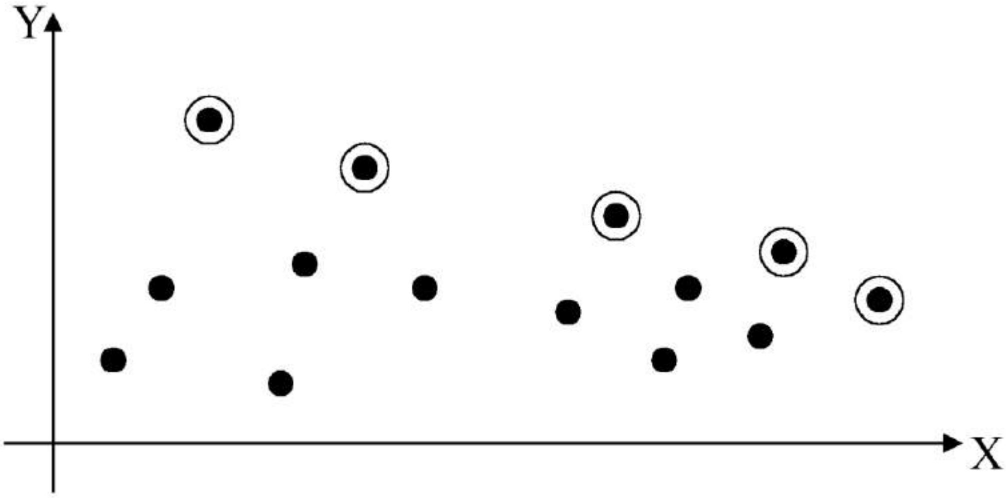
e.g. binary search

e.g. quick sort

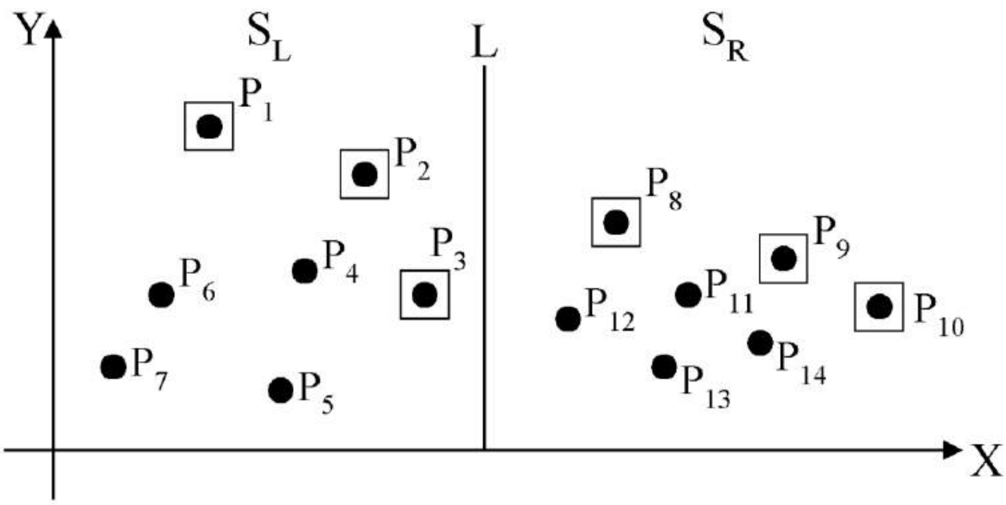
e.g. merge sort

- **2-D maxima finding problem**

Def: A point (x_1, y_1) dominates (x_2, y_2) if $x_1 > x_2$ and $y_1 > y_2$. A point is called a maxima if no other point dominates it.



Straightforward method:
compare every pair of points
time complexity: $O(n^2)$.



The maximal of S_L and S_R

Algorithm 5.1 A Divide-and-Conquer Approach to Find Maximal Points in the Plane

Input: A set of n planar points.

Output: The maximal points of S.

Step 1. If S contains only one point, return it as the maxima. Otherwise, find a line L perpendicular to the X-axis which separates the set of points into two subsets S_L and S_R, each of which consisting of n/2 points.

Step 2. Recursively find the maximal points of S_L and S_R.

Step 3. Project the maximal points of S_L and S_R onto L and sort these points according to their y-values. Conduct a linear scan on the projections and discard each of the maximal points of S_L if its y-value is less than the y-value of some maximal point of S_R.

time complexity:

$$T(n)=\begin{cases} 2T(n/2)+O(n)+O(n \log n) & , n > 1 \\ 1 & , n = 1 \end{cases}$$

Assume $n = 2^k$

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

Improvement:

(1)Step 3: Find the largest y-value of S_R.

time complexity:

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

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

(2)The sorting of y-values need be done only once(only one presorting).

No sorting is needed in Step3.

time complexity:

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

where

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

- **The closest pair problem**

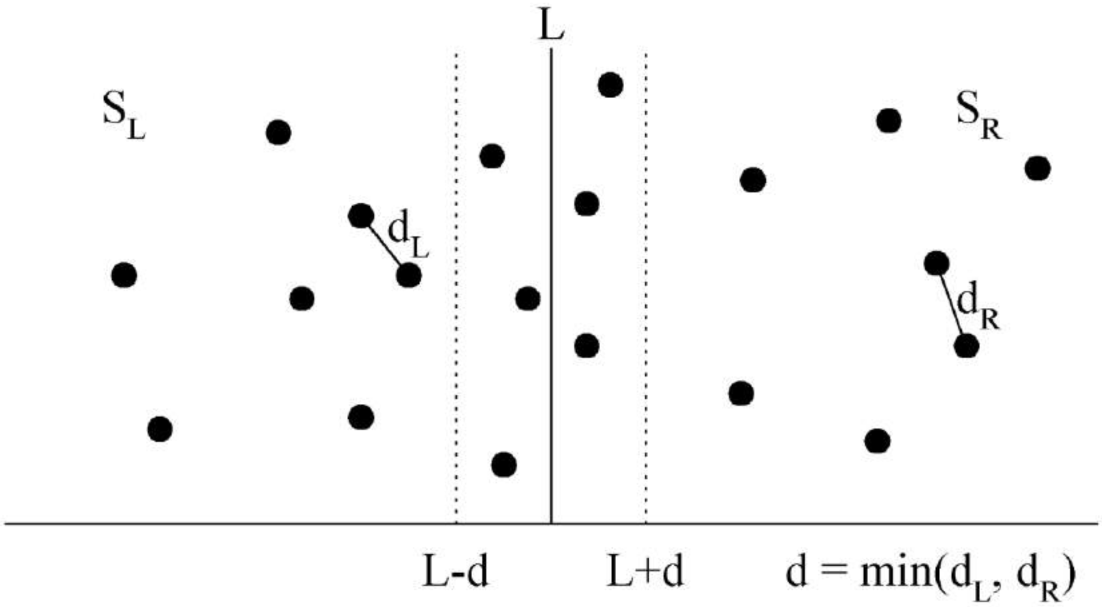
Given a set S of n points, find a pair of points which are closest together.

1-D version:

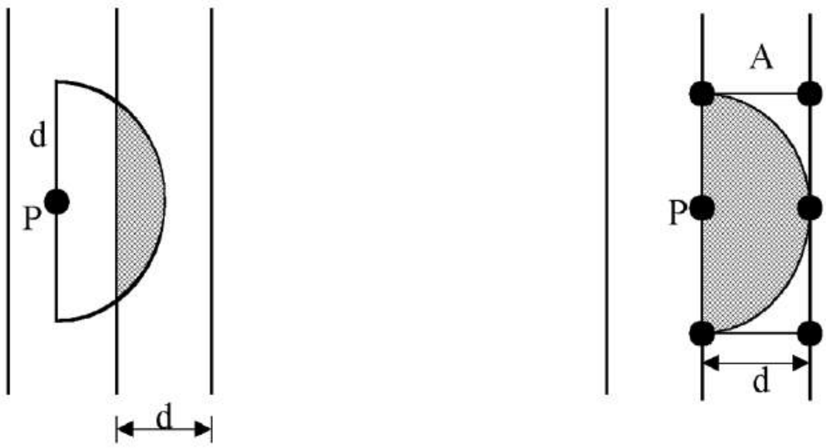
solved by sorting

time: O(n log n)

2-D version:



at most 6 points in A:



<u>Algorithm 5.2 A Divide-and-Conquer Algorithm to Solve the 2-Dimensional Closest Pair Problem</u>

<u>Input</u>: A set S of n points in the plane.

<u>Output</u>: The distance between two closest points.

Step 1. Sort points in S according to their y-values and x-values.

Step 2. If S contains only one point, return ∞ as its distance.

Step 3. Find a median line L perpendicular to the X-axis to divide S into two subsets, with equal sizes, S_L and S_R. Every point in S_L lies to the left of L and every point in S_R lies to the right of L.

Step 4. Recursively apply Step 2 and Step 3 to solve the closest pair problems of S_L and S_R. Let $d_L(d_R)$ denote the distance between the closest pair in SL(SR). Let $d = \min(d_L, d_R)$.

Step 5. Project all points within the slab bounded by L-d and L+d onto the line L. For a point P in the half-slab bounded by L-d and L, Let its y-value by denoted as y_P. For each such P, find all points in the half-slab bounded by L and L+d whose y-value fall within y_P+d and y_P-d. If the distance d′ between P and a point in the other half-slab is less than d, let d=d′. The final value of d is the answer.

time complexity: O(n log n)

Step 1: O(n log n)

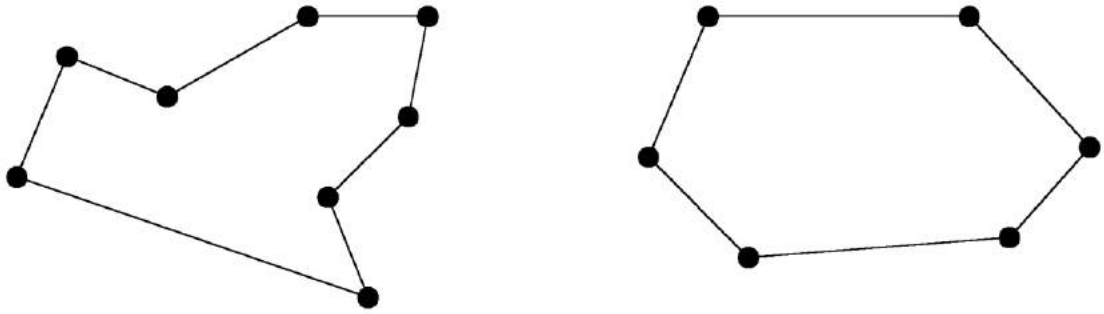
Steps 2~5:

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

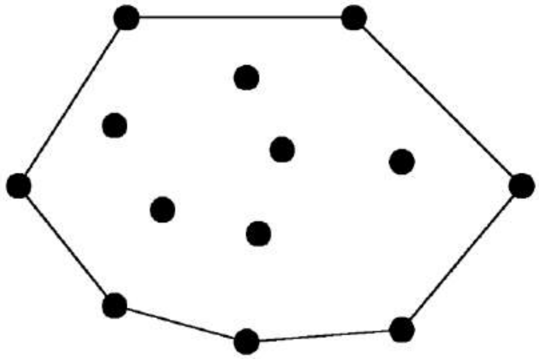
$\Rightarrow T(n) = O(n \log n)$

- **The convex hull problem**

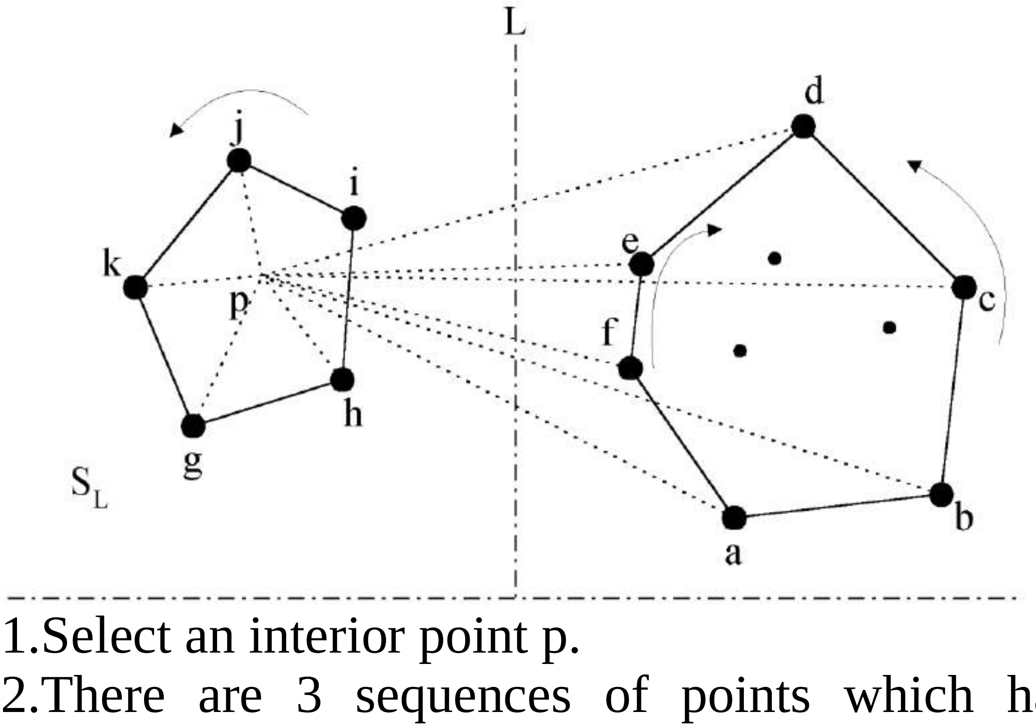
concave polygon: convex polygon:



The convex hull of a set of planar points is the smallest convex polygon containing all of the points.



the divide-and-conquer strategy to solve the problem:

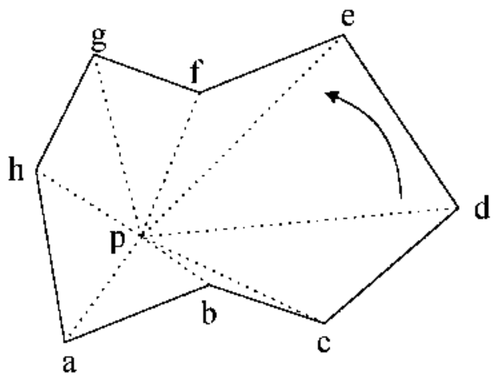


1. Select an interior point p.
2. There are 3 sequences of points which have increasing polar angles with respect to p.
   (1) g, h, i, j, k
   (2) a, b, c, d

(3) f, e

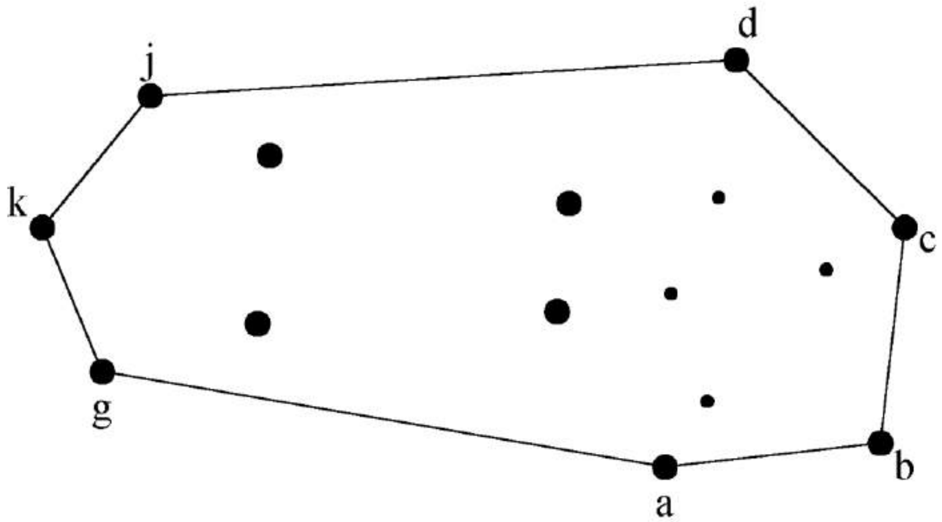
3.Merge these 3 sequences into 1 sequence:
g, h, a, b, f, c, e, d, i, j, k.

4.Apply <u>Graham scan</u> to examine the points one by one and eliminate the points which cause <u>reflexive angles</u>.



e.g. points b and f need to be deleted.

Final result:



<u>Algorithm 5.3 An Algorithm to Construct a Convex Hull Based Upon the Divide-and-Conquer Strategy</u>

<u>Input</u>: A set S of planar points

<u>Output</u>: A convex hull for S

<u>Step 1.</u> If S contains no more than five points, use exhaustive searching to find the convex hull and return.

<u>Step 2.</u> Find a median line perpendicular to the X-

axis which divides S into S_L and S_R; S_L lies to the left of S_R.

<u>Step 3.</u> Recursively construct convex hulls for S_L and S_R. Denote these convex hulls by Hull(S_L) and Hull(S_R) respectively.

<u>Step 4.</u> Find an interior point P of S_L. Find the vertices v_1 and v_2 of Hull(S_R) which divide the vertices of Hull(S_R) into two sequences of vertices which have increasing polar angles with respect to P. Without loss of generality, let us assume that v_1 has greater y-value than v_2. Then form three sequences as follows:

(a) Sequence 1: all of the convex hull vertices of Hull(S_L) in counterclockwise direction.

(b) Sequence 2: the convex hull vertices of Hull(S_R) from v_2 to v_1 in counter-clockwise direction.

(c) Sequence 3: the convex hull vertices of Hull(S_R) from v_2 to v_1 in clockwise direction.

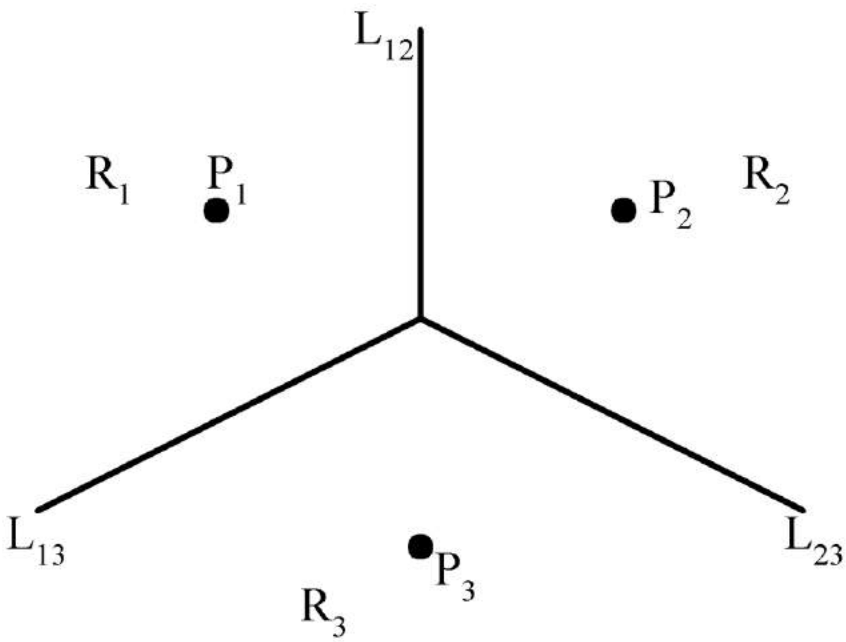
Merge these three sequences and conduct the Graham scan. Eliminate the points which are reflexive and the remaining points from the convex hull.

time complexity:

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

- **The Voronoi diagram problem**

e.g.



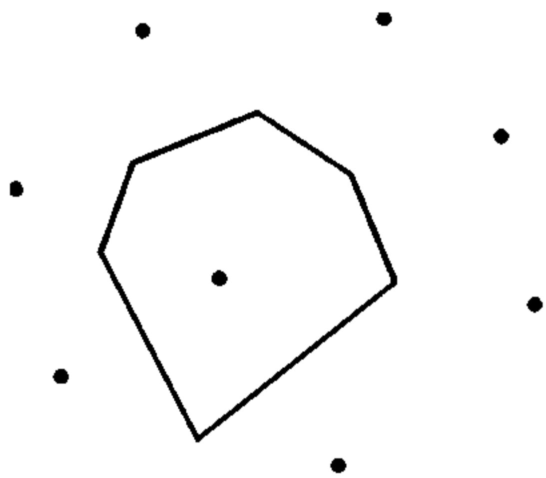
The Voronoi Diagram for Three Points

Each L_{ij} is perpendicular bisector of the line $\overline{P_i P_j}$.

<u>Def</u>: Given two points P_i, $P_j \in S$, let $H(P_i, P_j)$ denote the half plane containing P_i. The <u>Voronoi polygon</u> associated with P_i is defined as

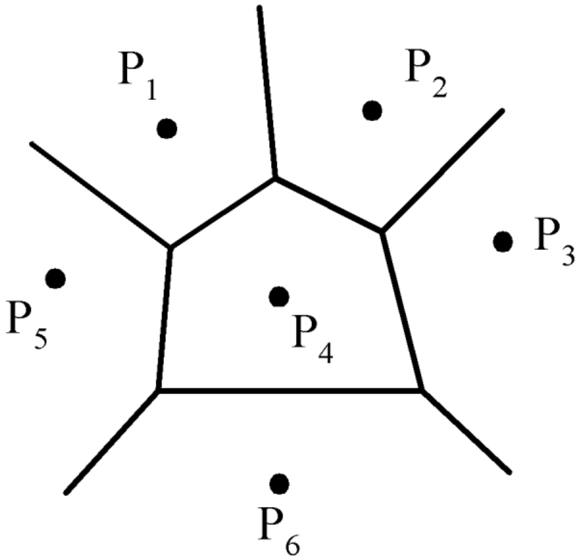
$$V(i) = \bigcap_{i \neq j} H(P_i, P_j)$$

e.g.



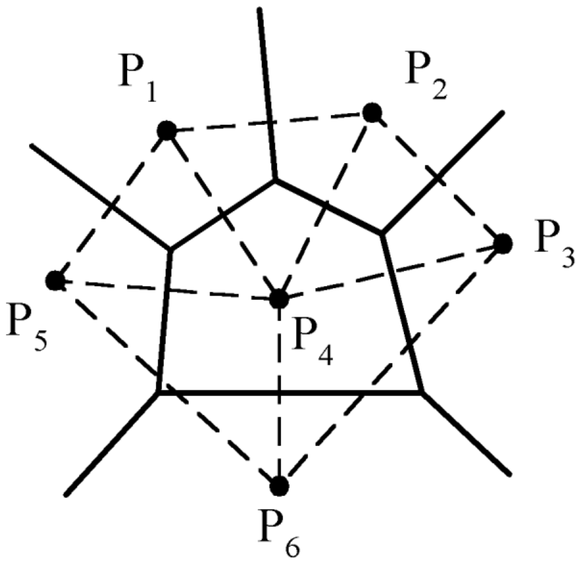
Given a set of n points, the <u>Voronoi diagram</u> consists of all the Voronoi polygons of these points.

e.g. A Voronoi diagram of 6 points:



The vertices of the Voronoi diagram are called Voronoi points and its segments are called Voronoi edges.

e.g. A Delaunay triangulation:



Algorithm 5.4 A Divide-and-Conquer Algorithm to Construct Voronoi Diagrams

Input: A set S of n planar points.

Output: The Voronoi diagram of S.

Step 1. If S contains only one point, return.

Step 2. Find a median line L perpendicular to the X-axis which divides S into S_L and S_R such that

$S_L(S_R)$ lies to the left(right) of L and the sizes of S_L and S_R are equal.

Step 3. Construct Voronoi diagrams of S_L and S_R recursively. Denote these Voronoi diagrams by $VD(S_L)$ and $VD(S_R)$.

Step 4. Construct a dividing piece-wise linear hyperplane HP which is the locus of points simultaneously closest to a point in S_L and a point in S_R. Discard all segments of $VD(S_L)$ which lie to the right of HP and all segments of $VD(S_R)$ that lie to the left of HP. The resulting graph is the Voronoi diagram of S.

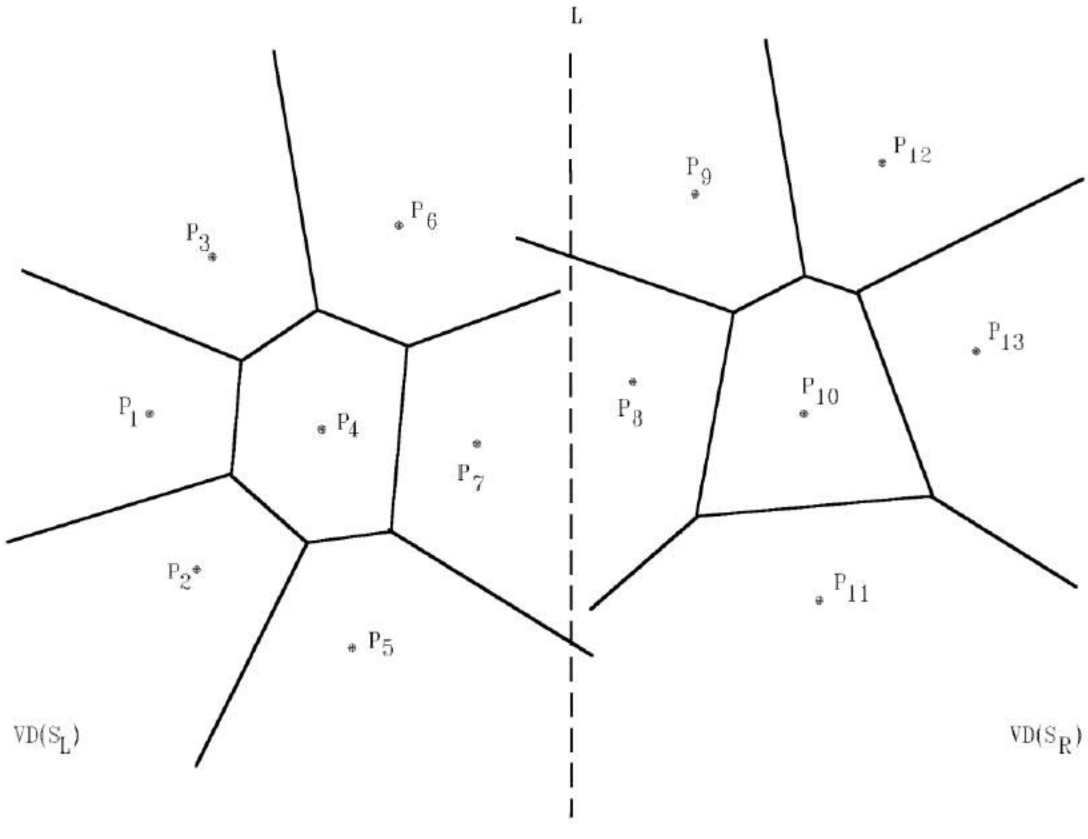


Fig. 5-17: Two Voronoi Diagrams After Step 2

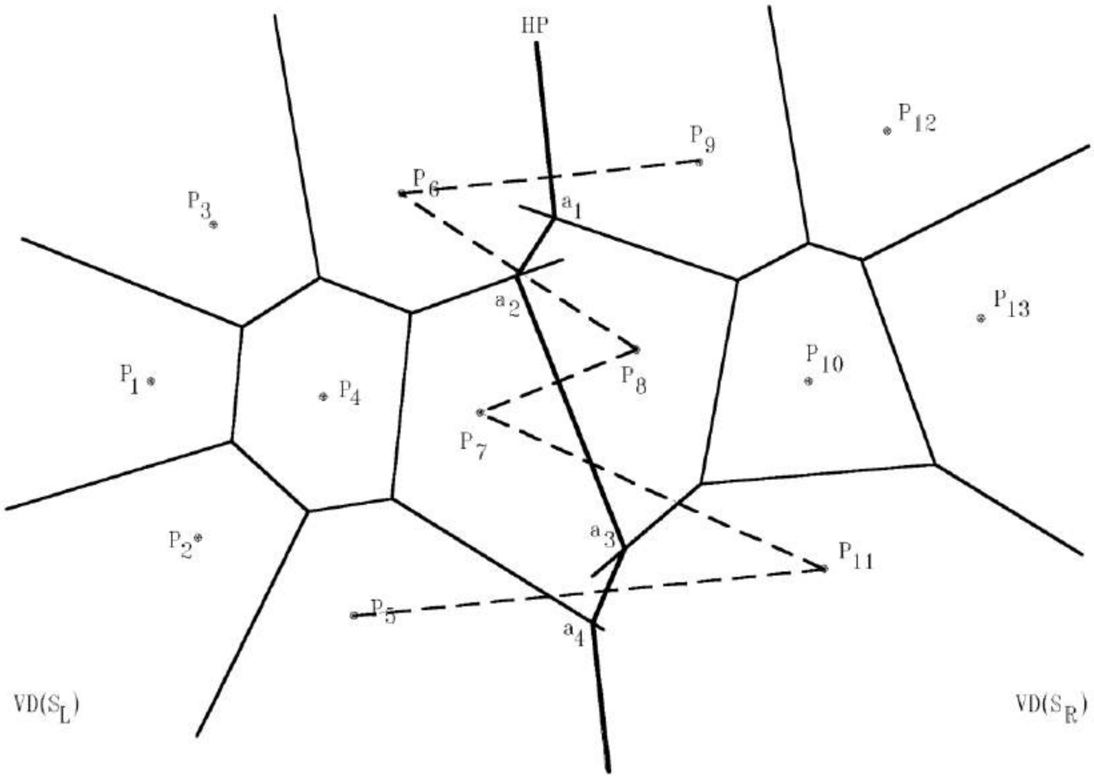


Fig. 5–18: The Piecewise Linear Hyperplane for the set of Points Shown in Fig. 5–17.

The Final Voronoi diagram:

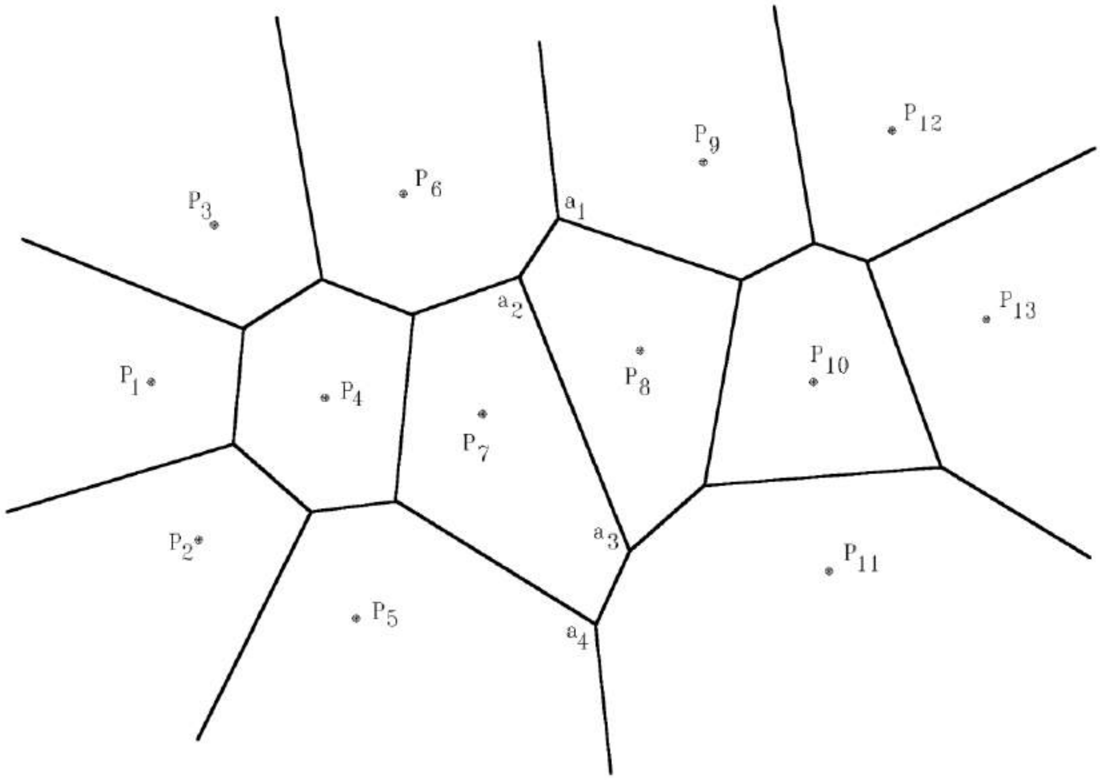


Fig. 5–19: The Voronoi Diagram of the Points in Fig. 5–17.

How to merge two Voronoi diagrams?

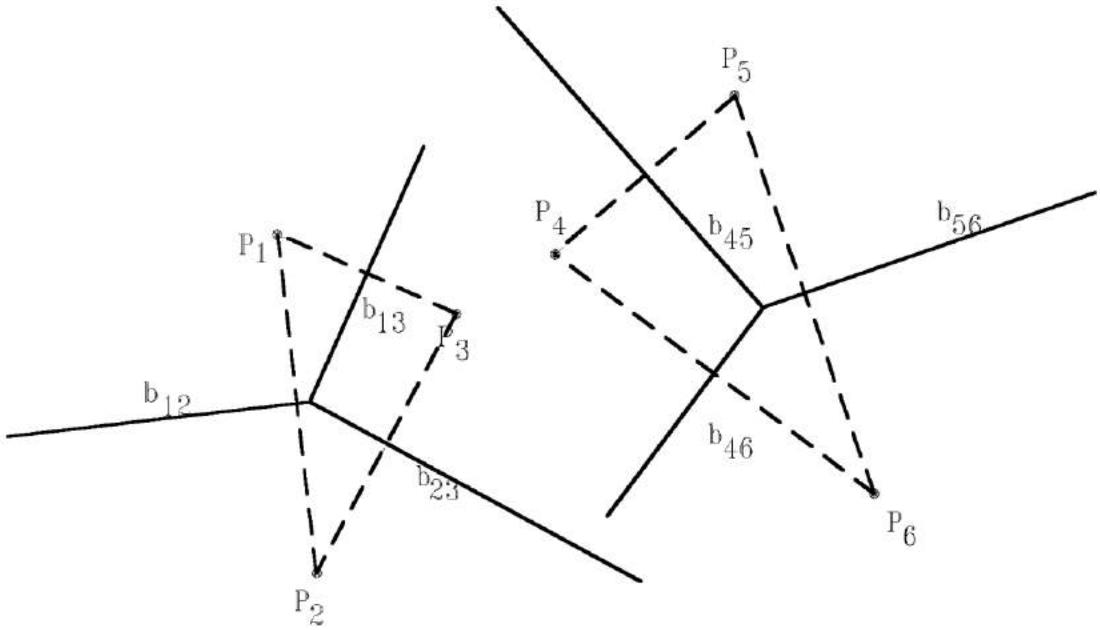


Fig 5-20: Another Case to Illustrate the Construction of Voronoi Diagrams

merging:

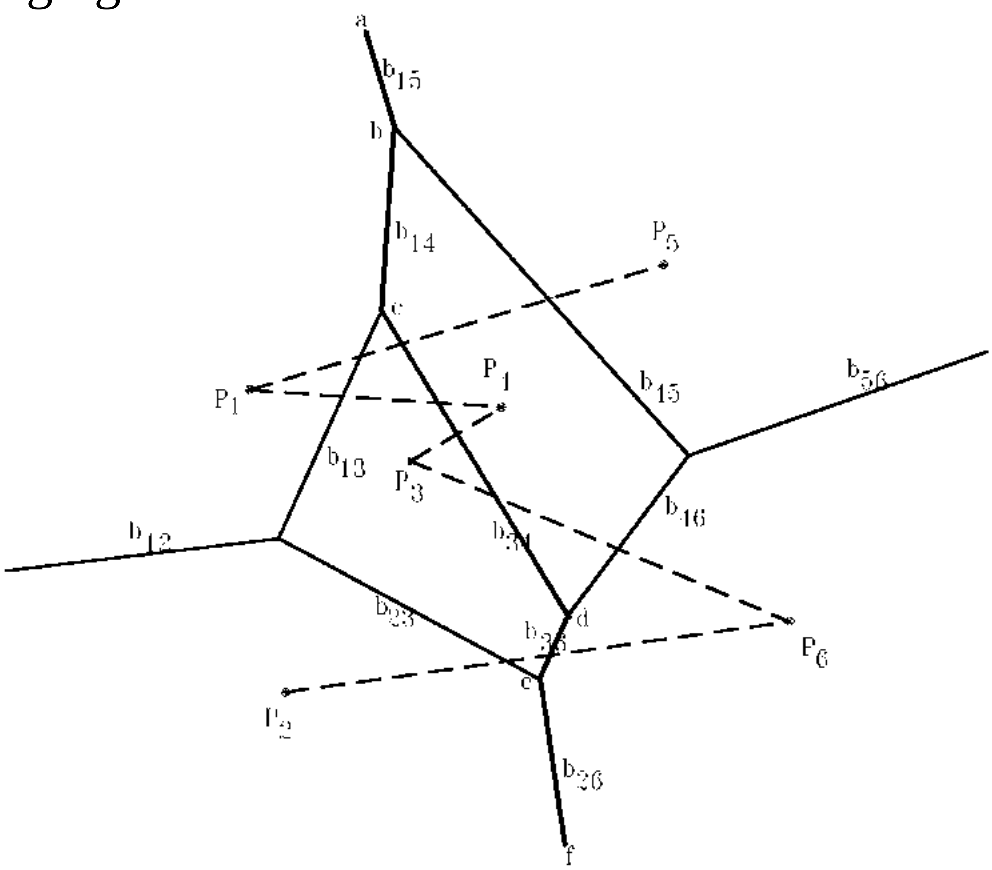


Fig 5-21: The Merging Step of Constructing a Voronoi Diagram

<u>Algorithm 5.5 An Algorithm which Merges Two</u>

Voronoi Diagrams into One Voronoi Diagram

Input: (a) S_L and S_R where S_L and S_R are divided by a perpendicular line L.
(b) $VD(S_L)$ and $VD(S_R)$.

Output: VD(S) where $S = S_L \cap S_R$

Step 1. Find the convex hulls of SL and SR. Let them be denoted as $Hull(S_L)$ and $Hull(S_R)$ respectively. (A special algorithm for finding a convex hull in this case will by given later.)

Step 2. Find segments $\overline{P_a P_b}$ and $\overline{P_c P_d}$ which join $HULL(S_L)$ and $HULL(S_R)$ into a convex hull (P_a and P_c belong to S_L and P_b and P_d belong to S_R.) Assume that $\overline{P_a P_b}$ lies above $\overline{P_c P_d}$. Let x = a, y = b, SG= $\overline{P_x P_y}$ and HP = $\varnothing$.

Step 3. Find the perpendicular bisector of SG. Denote it by BS. Let HP = HP $\cup$ {BS}. If SG = $\overline{P_c P_d}$, go to Step 5; otherwise, go to Step 4.

Step 4. The ray from $VD(S_L)$ and $VD(S_R)$ which BS first inter sects with must by a perpendicular bisector of either $\overline{P_x P_z}$ or $\overline{P_y P_z}$ for some z. If this ray is the perpendicular bisector of $\overline{P_y P_z}$, then let SG = $\overline{P_x P_z}$; otherwise, let SG = $\overline{P_z P_y}$. Go to Step 3.

Step 5. Discard the edges of $VD(S_L)$ which extend to the right of HP and discard the edges of $VD(S_R)$ which extend to the left of HP. The resulting graph is the Voronoi diagram of S =

$S_L \cup S_R$.

<u>Def</u>: Given a point P and a set S of points, the <u>distance</u> between P and S is the distance between P and P_i which is the nearest neighbor of P in S.

- The HP obtained from the above algorithm is the locus of points which keep equal distances to S_L and S_R.
- The HP is monotonic in y.
- # of edges of a Voronoi diagram $\leq 3n - 6$, where n is # of points.

  reasoning:

  (i) # of edges of a planar graph with n vertices $\leq 3n - 6$.

  (ii) A Delaunay triangulation is a planar graph.

  (iii) edges in Delaunay triangulation $\xleftrightarrow{1-1}$ edges in Voronoi diagram.
- # of Voronoi vertices $\leq 2n - 4$.

  reasoning:

  (i) Let F, E and V denote # of face, edges and vertices in a planar graph.

  Euler's relation: $F = E - V + 2$.

  (ii) In a Delaunay triangulation, $V = n$, $E \leq 3n - 6$

  $\Rightarrow F = E - V + 2 \leq 3n - 6 - n + 2 = 2n - 4$.
- After a Voronoi diagram is constructed, a convex hull can by found in O(n) time.

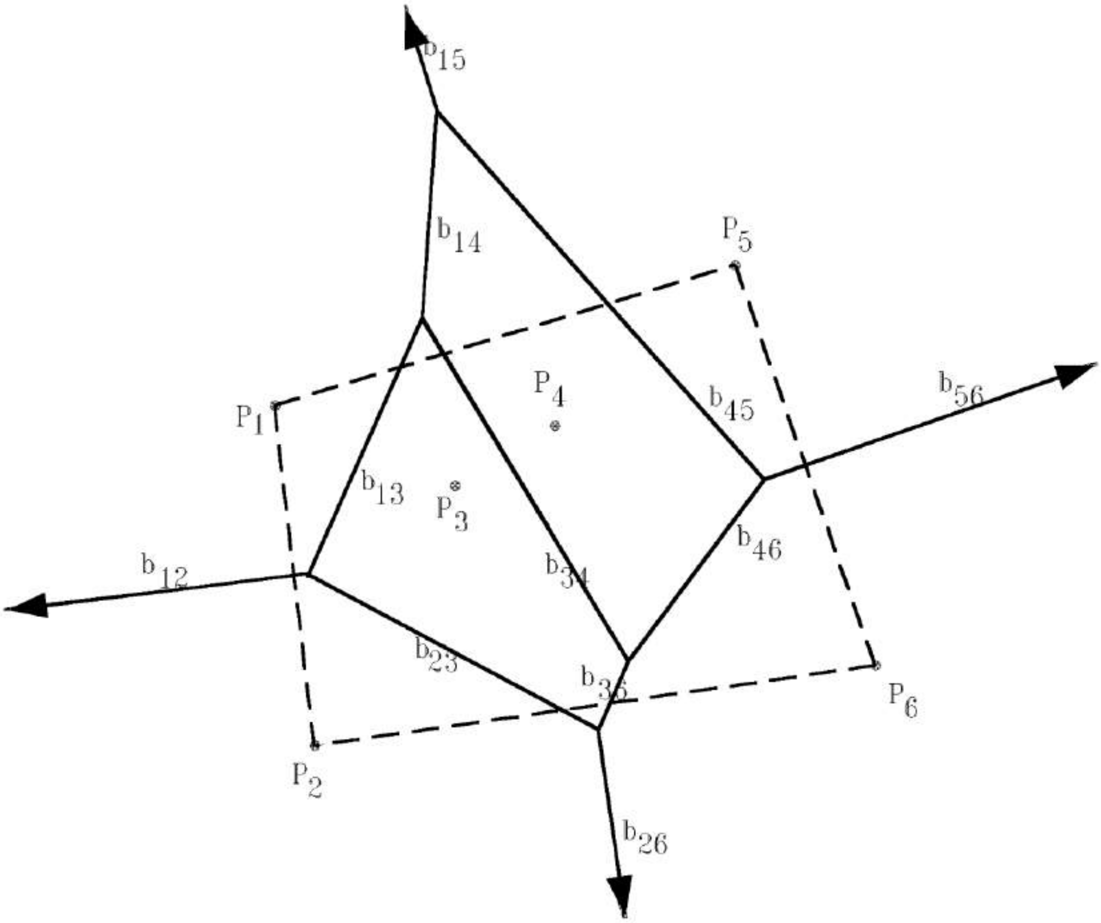


Fig. 5-25: Constructing a Convex Hull from a Voronoi Diagram

Step 1: Find an infinite ray by examining all Voronoi edges.

Step 2: Let P_i be the point to the left of the infinite ray. P_i is a convex hull vertex. Examine the Voronoi polygon of P_i to find the next infinite ray.

Step 3: Repeat Step 2 until we return to the Starting ray.

- time complexity for merging 2 Voronoi diagrams:
  Step 1: O(n)
  Step 2: O(n)
  Step 3 ~ Step 5: O(n)
  (at most 3n - 6 edges in VD(S_L) and VD(S_R) and

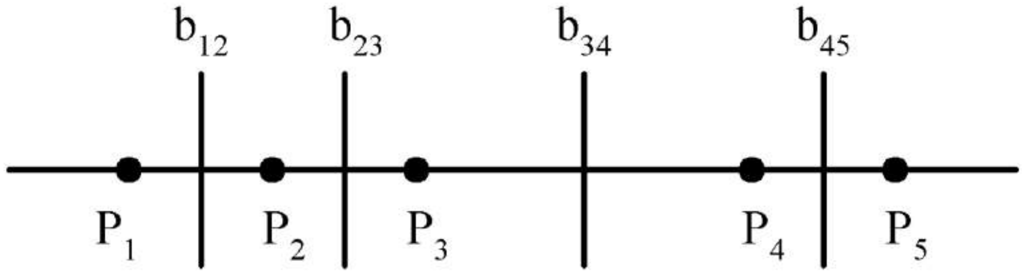
at most n segments in HP)

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

$= O(n \log n)$

- The lower bound of the Voronoi diagram problem is $\Omega(n \log n)$.

$\because$ sorting $\propto$ Voronoi diagram problem



The Voronoi Diagram for a Set of Points on a Straight Line

- Applications of the Voronoi diagrams
  - The Euclidean nearest neighbor searching problem.
  - The Euclidean all nearest neighbor problem.

- **The fast Fourier transform (FFT)**
  - Fourier transform:

$$A(f) = \int_{-\infty}^{\infty} a(t)e^{2\pi i f t}dt$$

  - inverse Fourier transform:

$$a(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} A(t)e^{-2\pi i f t}df$$

  - discrete Fourier transform(DFT):

    given $a_0, a_1, \cdots, a_{n-1}$

$$A_j = \sum_{0\le k\le n-1} a_k e^{2\pi ijk/n}, 0 \le j \le n-1$$

  - inverse DFT:

$$a_k = \frac{1}{n}\sum_{0\le j\le n-1} A_j e^{-2\pi ijk/n}, 0 \le k \le n-1$$

  - A straightforward method to calculate DFT requires $O(n^2)$ time.
  - DFT can be solved by the divide-and-conquer strategy.(FFT)

e.g. n =4

$A_0 = a_0 + a_1 + a_2 + a_3$

$A_1 = a_0 + a_1e^{2\pi i/4} + a_2e^{2\pi i(2)/4} + a_3e^{2\pi i(3)/4}$

$\quad = a_0 + a_1e^{\pi i/2} + a_2e^{\pi i} + a_3e^{3\pi i/2}$

$A_2 = a_0 + a_1e^{2\pi i(2)/4} + a_2e^{2\pi i(2)(2)/4} + a_3e^{2\pi i(2)(3)/4}$

$\quad = a_0 + a_1e^{\pi i} + a_2e^{2\pi i} + a_3e^{3\pi i}$

$A_3 = a_0 + a_1e^{2\pi i(3)/4} + a_2e^{2\pi i(3)(2)/4} + a_3e^{2\pi i(3)(3)/4}$

$\quad = a_0 + a_1e^{3\pi i/2} + a_2e^{3\pi i} + a_3e^{9\pi i/2}$

rewrite as:

$A_0 = a_0 + a_1 + a_2 + a_3$

$A_1 = a_0 + a_1e^{\pi i/2} + a_2e^{\pi i} + a_3e^{3\pi i/2}$

$A_2 = a_0 + a_1e^{\pi i} + a_2e^{2\pi i} + a_3e^{3\pi i}$

$A_3 = a_0 + a_1e^{3\pi i/2} + a_2e^{3\pi i} + a_3e^{9\pi i/2}$

another from:

$A_0 = a_0 + a_2 + (a_1 + a_3)$

$A_2 = a_0 + a_2e^{2\pi i} + (a_1e^{\pi i} + a_3e^{3\pi i})$

$\quad = a_0 + a_2 - (a_1 + a_3)$

When we calculate A_0, we shall calculate $(a_0 + a_2)$ and $(a_1 + a_3)$. Later, A_2 can be easily found.

Similarly,

$A_1 = (a_0 + a_2e^{\pi i}) + a_1e^{\pi i/2} + a_3e^{3\pi i/2}$

$A_3 = (a_0 + a_2e^{3\pi i}) + (a_1e^{3\pi i/2} + a_3e^{9\pi i/2})$

$\quad = (a_0 + a_2e^{\pi i}) - (a_1e^{\pi i/2} + a_3e^{3\pi i/2})$

e.g. $n = 8$, let $e^{2\pi i/8} = w = e^{\pi i/4}$

$(w^8 = 1,\ w^4 = -1)$

$A_0 = a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$

$A_1 = a_0+a_1w+a_2w^2+a_3w^3+a_4w^4 +a_5w^5+a_6w^6+a_7w^7$

$A_2 = a_0+a_1w^2+a_2w^4+a_3w^6+a_4w^8+a_5w^{10}+a_6w^{12}+a_7w^{14}$

$A_3 = a_0+a_1w^3+a_2w^6+a_3w^9+a_4w^{12}+a_5w^{15}+a_6w^{18}+a_7w^{21}$

$$\Downarrow$$

$A_0 = (a_0+a_2+a_4+a_6)+(a_1+a_3 +a_5 +a_7)$

$A_1 = (a_0+a_2w^2+a_4w^4+a_6w^6)+(a_1w+a_3w^3+a_5w^5+a_7w^7)$

$A_2 = (a_0+a_2w^4+a_4w^8+a_6w^{12})+(a_1w^2+a_3w^6+a_5w^{10}+a_7w^{14})$

$A_3 = (a_0+a_2w^6+a_4w^{12}+a_6w^{18})+(a_1w^3+a_3w^9+a_5w^{15} +a_7w^{21})$

Rewrite as:

$A_0 = B_0 + C_0$	$A_4 = B_0 - C_0$
$A_1 = B_1 + C_1$	$A_5 = B_1 - C_1$
$A_2 = B_2 + C_2$	$A_6 = B_2 - C_2$
$A_3 = B_3 + C_3$	$A_7 = B_3 - C_3$

We can apply the same method to calculate B_i's and C_i's.

$B_0 = a_0+a_2+a_4+a_6$

$B_2 = a_0+a_2w^4+a_4w^8+a_6w^{12}$

Rewrite as:

$B_0 = (a_0+a_4)+(a_2+a_6)$

$B_2 = (a_0+a_4w^8)+(a_2w^4+a_6w^{12})$

$\quad = (a_0+a_4)-(a_2+a_6)$

Similarly,

$B_1 = a_0+a_2w^2+a_4w^4+a_6w^6$

$\quad = (a_0+a_4w^4) + (a_2w^2+a_6w^6)$

$B_3 = a_0+a_2w^6+a_4w^{12}+a_6w^{18}$

$\quad = (a_0+a_4w^{12}) + (a_2w^6+a_6w^{18})$

$\quad = (a_0+a_4w^4) - (a_2w^2+a_6w^6)$

In general, let $w = e^{2\pi i/n}$ (assume n is even)

$(w^n = 1, w^{n/2} = -1)$

$A_j \quad = a_0+a_1w^j+a_2w^{2j}+\cdots+a_{n-1}w^{(n-1)j}$

$\quad = \{a_0+a_2w^{2j}+a_4w^{4j}+\cdots+a_{n-2}w^{(n-2)j}\} +$

$\quad\quad \{a_1w^j+a_3w^{3j}+\cdots+a_{n-1}w^{(n-1)j}\}$

$\quad = B_j + C_j$

$A_{j+n/2} \quad = a_0+a_1w^{j+n/2}+a_2w^{2j+n}+a_3w^{3j+3n/2}+\cdots+$

$\quad\quad a_{n-1}w^{(n-1)j+(n(n-1)/2)}$

$\quad = a_0-a_1w^j+a_2w^{2j}-a_3w^{3j}+\cdots+a_{n-2}w^{(n-2)j}-a_{n-1}w^{(n-1)j}$

$\quad = B_j - C_j$

Algorithm 5.6 A Fast Fourier Transform Algorithm Based Upon the Divide-and-Conquer Strategy

Input: $a_0, a_1, \cdots, a_{n-1}, n = 2^k$

Output: A_j, j=0, 1, 2, $\cdots$, n-1

where $A_j = \sum_{0 \le k \le n-1} a_k e^{2\pi ijk/n}$

Step 1. If n=2, compute

$A_0 = a_0 + a_1$,

$A_1 = a_0 - a_1$, and

return.

Step 2. Divide each A_j, $0 \le j \le n/2 - 1$ into two sequences:

O_j and E_j, where $O_j(E_j)$, consists of odd-numbered(even-numbered) terms of A_j.

Step 3. Recursively calculate the sums of terms in O_j and E_j. Denote the sum of terms of O_j and E_j by B_j and C_j respectively.

Step 4. Compute A_j by the following formual:

$A_j = B_j + C_j$ for $0 \le j \le n/2 - 1$

$A_{j+n/2} = B_j - C_j$ for $0 \le j \le n/2 - 1$.

time complexity:

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

- **Strassen's matrix multiplicaiton**

Let A, B and C by $n \times n$ matrices

$C = AB$

$$C(i, j) = \sum_{1 \le k \le n} A(i, k)B(k, j)$$

- The straightforward method to perform a matrix multiplication requires $O(n^3)$ time.
- The divide-and-conquer strategy:

$C = AB$

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

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$

$C_{12} = A_{11}B_{12} + A_{12}B_{22}$

$C_{21} = A_{21}B_{11} + A_{22}B_{21}$

$C_{22} = A_{21}B_{12} + A_{22}B_{22}$

time complexity:

$$T(n) = \begin{cases} b & , n \le 2 \\ 8T(n/2)+cn^2 & , n > 2 \end{cases}$$

(# of additions: n^2)

$\Rightarrow T(n) = O(n^3)$

- Strassen's method:

$P = (A_{11} + A_{22})(B_{11} + B_{22})$

$Q = (A_{21} + A_{22})B_{11}$

$R = A_{11}(B_{12} - B_{22})$

$S = A_{22}(B_{21} - B_{11})$

$T = (A_{11} + A_{12})B_{22}$

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

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

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

$C_{12} = R + T$
$C_{21} = Q + S$
$C_{22} = P + R - Q + U$

7 multiplications and 18 additions or subtractions.

time complexity:

$$T(n) = \begin{cases} b & , n \le 2 \\ 7T(n/2)+an^2 & , n > 2 \end{cases}$$

$$\begin{aligned}
\Rightarrow T(n) &= an^2 + 7T(n/2) \\
&= an^2 + 7(a(n/2)^2 + 7T(n/4)) \\
&= an^2 + (7/4)an^2 + 7^2T(n/4) \\
&= \cdots \\
&\quad \vdots \\
&= an^2(1 + 7/4 + (7/4)^2+\cdots+(7/4)^{k-1}+7^kT(1)) \\
&\le cn^2(\tfrac{7}{4})^{\log_2 n} + 7^{\log_2 n}, \text{ c is a constant} \\
&= c n^{\log_2 4+\log_2 7-\log_2 n} + n^{\log_2 7} \\
&= O(n^{\log_2 7}) \\
&\cong O(n^{2.81})
\end{aligned}$$