1.

We first compute the Voronoi diagram of point set B ($V.D.(B)$), and get a subdivision of the plane with $O(n)$ complexity in $O(n \log n)$ time. Then for each point $a_i$ of A, we only have to decide which cell it lies in. The point $b_j$ associated with the cell is the nearest neighbor of $a_i$. To decide which cell $a_i$ lies in is actually a point location problem. Using the method in Chapter 6, we can create the trapezoidal map of $V.D.(B)$ in $O(n \log n)$ expected time, and the expected query time is $O(\log n)$.

In general, to find the location cells for all points in A, the total time is bounded by:

$O(n \log n) + O(n \log n) + n \cdot O(\log n) = O(n \log n)$.

7.11

We can compute the Voronoi diagram of the point set, and get the Delaunay triangulation ($D.T.$) in $O(n \log n)$ time. For each point, its closest neighbor must be one of the points that have an edge to it. This can be proved by contradiction using Theorem 9.6. So for each point $i$, we only have to find its closest neighbor. This can be done in Degree($i$) time. Thus the total checking time is the total degrees of all vertices, which is twice the number of edges in $D.T$. As the complexity of $D.T.$ is bounded by $O(n)$, the checking can be done in $O(n)$ time. In general therefore, the algorithm can find the nearest neighbor for each point in $O(n \log n)$ time.

8.1

Given point $p : (p_x, p_y)$ and line $l : y = mx + b$ in primal space, if $p$ lies above $l$, we have:

$p_y > m \cdot p_x + b$, so: $m \cdot p_x - p_y + b < 0$.

In the dual space, we have $p^*: y = p_x \cdot x - p_y$, $l^*: (m,-b)$, so:
\[ -b - (p_x \cdot m - p_y) = -(m \cdot p_x - p_y + b) > 0 \]

Hence \( l^* \) lies above \( p^* \), and order preserving holds.

Similarly, if \( p \) lies on \( l \), we have \( m \cdot p_x - p_y + b = 0 \); in the dual space,
\[ -b - (p_x \cdot m - p_y) = -(m \cdot p_x - p_y + b) = 0 \]

Hence \( l^* \) lies on \( p^* \), and the incidence preserving holds.

8.4

To decide a rectangular box, we only have to find the top (T), bottom (B), left (L) and right (R) intersection points in the primal space (as shown in the figure). Suppose the line set is \( \{l_1, l_2, \ldots, l_n\} \), and the intersection points set is \( \{p_1, p_2, \ldots, p_m\} \), in which \( m \) is the total number of intersections. So in the dual space we’ll get a point set \( \{l_1^*, l_2^*, \ldots, l_n^*\} \), and a line set \( \{p_1^*, p_2^*, \ldots, p_m^*\} \)

First consider the top point \( p_T : (x_T, y_T) \), which lies on or above \( \{l_1, l_2, \ldots, l_n\} \). So in the dual space, according to Observation 8.3, \( \{l_1^*, l_2^*, \ldots, l_n^*\} \) must all lie on or above \( p_T^* \). Thus \( p_T^* \) must be one of
edges in the lower convex hull. Further more, as $p_T^* : y = x_T \cdot x - y_T$, $p_T^*$ must be the one that has the lowest y-intersect ($-y_T$). Given n lines in primal, we have n points in dual. So the complexity of convex hull in dual is bounded by O(n). By checking its edges on lower hull one-by-one, we can find $p_T^*$ in O(n) time, thus find the top most intersection point in the primal space.

Situation for the bottom point $p_B^* : (x_B, y_B)$ is similar, and we can also find it in O(n) time in dual.

Next consider the left point $p_L^* : (x_L, y_L)$, which lies on or left to $\{l_1, l_2, \ldots, l_n\}$. Then by only a few algebra operations, we can decide if $\{l_1^*, l_2^*, \ldots, l_n^*\}$ all lies left or right to $p_T^*$, similarly to Observation 8.3. Thus $p_L^*$ must be on the left or right hull ( $p_R^*$ is opposite accordingly). Further more, $p_L^*$ must be the edge that has the lowest slope $x_L$, and $p_L^*$ must be the edge that has the largest slope $x_T$. This can also be decided on both left and right hull one-by-one, in O(n) time.

To summarize, we can find the four critical points by checking the convex hull in dual space in O(n) time. To construct the convex hull of the n “dual points”, we can use whatever methods given in Chapter 1, which is done in O(nlogn) time. Thus in general, this algorithm can find a rectangular box in O(nlogn) time.

8.7

Given two sets of point $R = \{p_{R1}, p_{R2}, \ldots, p_{Rn}\}$ and $B = \{p_{B1}, p_{B2}, \ldots, p_{Bn}\}$, we are looking for a line $l : y = mx + b$ that can separate them. This is a set of linear constraint:

$$\begin{cases} m \cdot x_{Ri} - y_{Ri} + b \leq 0 \\ m \cdot x_{Bi} - y_{Bi} + b \geq 0 \end{cases} \text{ or } \begin{cases} m \cdot x_{Ri} - y_{Ri} + b \geq 0 \\ m \cdot x_{Bi} - y_{Bi} + b \leq 0 \end{cases} \quad \forall i \in \{1, 2, \ldots, n\}$$

In the dual space, point sets become line sets:

$$R^* = \left\{ \begin{array}{l} R_1^* : y = x_{R1} \cdot x - y_{R1} \\ R_2^* : y = x_{R2} \cdot x - y_{R2} \\ \vdots \\ R_n^* : y = x_{Rn} \cdot x - y_{Rn} \end{array} \right\}, \text{ and } B^* = \left\{ \begin{array}{l} B_1^* : y = x_{B1} \cdot x - y_{B1} \\ B_2^* : y = x_{B2} \cdot x - y_{B2} \\ \vdots \\ B_n^* : y = x_{Bn} \cdot x - y_{Bn} \end{array} \right\}$$
Line $l$ becomes a point: $l^* = (m, -b)$.

Thus according to Observation 8.3, the linear constraint becomes:
\[
\begin{cases}
  x_{R_i} \cdot m - y_{R_i} + b \geq 0 \\
  x_{B_i} \cdot m - y_{B_i} + b \leq 0
\end{cases}
\]

or
\[
\begin{cases}
  x_{R_i} \cdot m - y_{R_i} + b \leq 0 \\
  x_{B_i} \cdot m - y_{B_i} + b \geq 0
\end{cases}
\]

This is actually to find a feasible region for $(m, b)$ given the $2n$ linear constraint above. We can solve it with the randomized linear programming algorithm in Section 4.4, and the expected running time is bounded by $O(n)$.

9.11

(a) In planar Euclidean space, a set $P$ of $n$ points has $n^2/2$ possible edges between one another. Prove by contradiction. Suppose edge $pq \in EMST(P)$, but $pq \not\in D.G.(P)$. Then according to Theorem 9.6 (ii), there must exist a point $r$ inside the circle with $pq$ as its center. Now let’s look at the closed loop defined by the triangle $pqr$. Apparently, $pq$ is larger than $pr$ and $rq$. However, according to the cycle property of MST [1][2], as $pq$ is the largest edge in $pqr$, it cannot exist in EMST($P$). This is a contradiction to our assumption. Thus $pq \in D.G.(P)$, and $EMST(P) \subseteq D.G.(P)$

(b) According to Theorem 7.10, we can calculate the Voronoi Diagram, and D.G. correspondingly in $O(n\log n)$ time, and we’re guaranteed that the complexity of D.G. is bounded in $O(n)$ space. Given the D.G. graph, with $O(n)$ vertices and $O(n)$ edges, we can use either Kruskal’s Alg. Or Prim’s Alg. [2] to find EMST($P$) in $O(n\log n)$ time.

In general, this algorithm runs in $O(n\log n)+ O(n\log n)= O(n\log n)$ time.
9.13

(a) Given a set of points \( P \), for any edge \( e \) that belongs to its Gabriel graph, suppose the two endpoints of \( e \) are \( p, q \). According to definition, the circle \( C \) with diameter \( \overline{pq} \) doesn’t contain any other point of \( P \). Thus we find a closed disc that contains \( p, q \) on its boundary and doesn’t contain any other point of \( P \). According to Theorem 9.6 (Text book, Page 198), \( e \) must belong to \( \text{D.G.}(P) \).

(b) Given a set of points \( P \), its Delaunay graph \( \text{D.G.} \) (considering general position case, such that \( \text{D.G} \) is a Delaunay triangulation), and two points \( p, q \) from it, which form an edge of \( \text{D.G.} \), let’s prove it from two directions:

\( \overline{pq} \) is a Gabriel edge \( \Rightarrow \overline{pq} \) intersect its dual Voronoi edge:

Prove by contradiction. There’s always another point \( r \), such that the triangle \( pqr \) belongs to \( \text{D.G.} \). Thus Voronoi egdes at the local area of can be shown below (red lines):

Suppose \( \overline{pq} \) doesn’t intersect its dual Voronoi edge \( \overline{uv} \), \( \overline{uv} \) must end before hitting \( \overline{pq} \). So there must be other vertex \( s \) from \( P \), which exists on the circle centering at \( v \), and crosses \( p, q \). If \( s \) is above \( \overline{pq} \) (shown by \( s1 \) in the figure), it must lie inside the circle with \( \overline{pq} \) as diameter, which contradict the definition of Gabriel graph; if \( s \) is below \( \overline{pq} \) (shown by \( s2 \) in the figure), it will lie inside the circle defined by \( pqr \), which contradict the feature of Delaunay graph (Theorem 9.6).

Both cases render a contradiction, which means the assumption is incorrect. Therefore, \( \overline{pq} \) must
intersect its dual Voronoi edge.

\[ pq \] is a Gabriel edge \( \iff \) \( pq \) intersect its dual Voronoi edge:

Similarly, we consider the Delaunay triangle \( pqr \) and its local Voronoi edges (shown below):

\[ pqr \] defines a circle centered at \( u \), and there must be another point \( s \) on the circle centered at \( v \) and cross both \( p \) and \( q \). Further more, \( s \) must lies above \( pq \). Otherwise, it will lie inside the circle defined by triangle \( pqr \). Thus \( s \) and \( r \) are the two neighboring vertices that shares common edge \( pq \) in the Delaunay triangulation. So we only have to check whether \( s \) or \( r \) lies inside the circle \( C \) with \( pq \) as the diameter. Apparently, \( arc-psq \) and \( arc-qrq \) both fully contains \( C \), so there is no point inside \( C \). Therefore, \( pq \) defines a Gabriel edge.

\((c)\) According to Theorem 7.10, we can calculate the Voronoi Diagram, and \( D.G. \) correspondingly in \( O(n \log n) \) time, and we’re guaranteed that the complexity of \( D.G. \) is bounded in \( O(n) \) space.

Then suppose we check an edge \( pq \) of \( D.G. \) if other vertices of \( P \) are inside the circle with \( pq \) as the diameter. **We only have to check the vertices that form triangles with \( pq \), because other vertices are guaranteed to be outside the circle. Otherwise, it will violate Theorem 9.6 (i).** Further more, as \( D.G. \) is planar, \( pq \) can be shared by at most 2 faces. Thus the checking procedure is bounded by
$2=O(1)$ time.

In this way, we remove an edge if there's point lying inside its circle, keep it otherwise. We perform this checking & updating procedure for all the $O(n)$ edges one-by-one. Finally we can get the Gabriel graph in $O(n)$ time.

In general therefore, the whole algorithm requires $O(n\log n)+O(n)= O(n\log n)$ time.

References