2. First, we show that the set of points in the plane that are closer to site s_i than to site s_j is a halfplane bounded by the perpendicular bisector of the segment s_is_j. This set is $\{p \mid d(p, s_i) \leq d(p, s_j)\}$. Let $p = (x, y)$, $s_i = (x_i, y_i)$, and $s_j = (x_j, y_j)$. The condition $d(p, s_i) \leq d(p, s_j)$ is then,

$$\begin{aligned} & \sqrt{(x-x_i)^2+(y-y_i)^2} \leq \sqrt{(x-x_j)^2+(y-y_j)^2} \\ \Leftrightarrow \quad & x_i^2 - x_j^2 - 2(x_i - x_j)x + y_i^2 - y_j^2 - 2(y_i - y_j)y \leq 0, \end{aligned}$$

which imposes a linear condition on (x, y). Notice that the set of points (x, y) that satisfy this condition with equality includes the midpoint of the segment s_is_j and is perpendicular to the vector $s_j - s_i$.

The halfplane defined by this perpendicular bisector is convex: If $p = (x, y)$ satisfies $ax+by \leq c$ and $p' = (x', y')$ satisfies $ax' + by' \leq c$ then $(1-t)p + tp'$ also satifies it, since

$$a((1-t)x + tx') + b((1-t)y + ty') = (1-t)(ax + by) + t(ax' + by') \leq (1-t)c + tc = c.$$

A Voronoi region R_i for site s_i is the intersection of the halfplanes defined by s_i and s_j for all $j = 1..n$. Since the intersection of convex sets is convex, R_i is convex.

4. The site s_i must lie in the interval $[m_{i-1}, m_i]$ and $m_i - s_i = s_{i+1} - m_i$ for all $i = 2..n-1$. For example, the possible locations of s_2 are in the interval $[m_1, m_2]$, and the location of s_3 must be the same distance from m_2 as s_2 (and to the right of m_2). So s_3 must lie in the intersection of $[m_2, m_3]$ and $m_2 + [m_2 - m_2, m_2 - m_1] = [m_2, 2m_2 - m_1]$. In general, if we know s_i lies in the interval $[a, b]$, then s_{i+1} must lie in the intersection of $[m_i, m_{i+1}]$ and $m_i + [m_i - b, m_i - a]$. If this intersection is empty, there is no possible location we can place s_{i+1}. Also, if the interval where we can place s_n is not empty, then we can place s_n in this interval and determine the positions of $s_{n-1}, s_{n-2}, \ldots, s_1$ in that order.

The algorithm starts with $[a, b] = [m_1, m_2]$ and $i = 2$. It updates the interval incrementally (as above) until either the interval becomes empty (in which case it says NO) or it reaches $i = n$ (in which case it says YES). The algorithm takes $O(n)$ time.

3. Let p and q be the closest pair of points of different color. Consider the circle with diameter $\overline{pq}$. This circle cannot contain a point $r \in S$ since $\overline{pr}$ and $\overline{rq}$ are both shorter than $\overline{pq}$, and one of them connects points of different color (contradicting the fact that p and q are the closest such pair). The existence of this empty circle implies $\overline{pq}$ is a Delaunay edge.

4. Let p be the lowest (and leftmost in case of ties) point in $P \cup Q$. We can find p in linear time. Assume it is in P (otherwise just exchange the names "P" and "Q"). The idea is to obtain the points of $(P \cup Q) - \{p\}$ in sorted angular order around p in linear time and then run Graham's Scan (without the angular sort step) to get the hull of $P \cup Q$ in linear time. The points of $P - \{p\}$ are already in angular order around p. Find the two tangents from the point p to the hull Q (taking linear time). Let q_1 and q_2 be the extreme CW and CCW tangent points in Q. Let Q_1 be the list of points in Q in CCW order from q_1 to q_2 (excluding q_2). Let Q_2 be the list of points in Q in CW order from q_1 to q_2 (excluding q_1). Both Q_1 and Q_2 are in sorted angular order around p. Merge $P - \{p\}$, Q_1, and Q_2 to form the sorted list of points in $(P \cup Q) - \{p\}$. The merging can be done in linear time just as in MergeSort.

5. Let $\overline{p_ip_j}$ realize the diameter of P. Consider the line ℓ perpendicular to $\overline{p_ip_j}$ that passes through p_i. This line is tangent to the circle centered at p_j with radius $d(p_i, p_j)$ (the distance from p_i to p_j). Every point $p_k \in P$ lies inside this circle since $d(p_j, p_k) \leq d(p_i, p_j)$. Thus ℓ is a strict supporting line for p_i, which implies p_i is a vertex on the boundary of the convex hull. The same argument applies to p_j.

The affine hull of two distinct points in the plane is the line through the two points. For three or more points in the plane, its the entire plane or, if the points all lie on a line, it is the line through the points. In 3D, the affine hull of two points is again the line through the two points. For three non-colinear points, its the plane containing the three points. For four or more non-planar points, its the entire space.

We show how to solve +ELEMENT UNIQUENESS using any algorithm for MAXDEPTH in only $O(n)$ additional time. Given input $x_1, x_2, \ldots, x_n$ to +ELEMENT UNIQUENESS, construct a set S of $4n$ points: $S = \{(x_i, 0), (0, x_i), (-x_i, 0), (0, -x_i) \mid \text{ for } 1 \leq i \leq n\}$. If the algorithm for MAXDEPTH outputs $n-1$ on input S, then output YES, otherwise output NO.

The correctness of the reduction relies on showing that the max depth of S is $n-1$ if and only if the n elements are unique. If $x_1, x_2, \ldots, x_n$ are unique then every convex hull in the sequence of nested convex hulls has only four points on its boundary from the set S, and therefore the max depth is $n-1$. If $x_i = x_j$ for some $i \neq j$, then the convex hull in the sequence that contains $(x_i, 0), (0, x_i), (-x_i, 0), (0, -x_i)$ also contains $(x_j, 0), (0, x_j), (-x_j, 0), (0, -x_j)$, so the max depth is at most $n-2$.

The running time of this algorithm for +ELEMENT UNIQUENESS is $O(n) + T_M^*(4n)$ where $T_M^*(4n)$ is the fastest running time of an algorithm solving MAXDEPTH on inputs of size $4n$. Since every algorithm for +ELEMENT UNIQUENESS takes time $\Omega(n \log n)$, this implies that $T_M^*(4n) \in \Omega(n \log n) - O(n)$ which implies $T_M^*(n) \in \Omega(n \log n)$.