Problem 1. Recall that, in discussing hierarchical clustering, we introduced 3 distance metrics on two sets of points: min, max, and mean. Let $S_1 = \{a, c\}$ and $S_2 = \{b, d\}$. What is the distance between $S_1$ and $S_2$ under those three metrics, respectively (assuming that the distance of two points is calculated by Euclidean distance)?

Answer. 
Min: $\sqrt{2}$, as is the distance between $a$ and $b$. 
Max: $\sqrt{17}$, as is the distance between $a$ and $d$. 
Mean: $(\sqrt{2} + \sqrt{17} + 2 + \sqrt{5})/4$, as is the average of $dist(a, c)$, $dist(a, d)$, $dist(c, b)$ and $dist(c, d)$.

Problem 2. Show the dendrogram returned by the Agglomerative algorithm under the min and max metrics, respectively.

Answer. 
Min. At the beginning of the algorithm, each point is regarded as a singleton cluster. In other words, there are 4 clusters, whose mutual distances are given by:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>-</td>
<td>$\sqrt{2}$</td>
<td>$\sqrt{10}$</td>
<td>$\sqrt{17}$</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td>2</td>
<td>$\sqrt{13}$</td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td>$\sqrt{5}$</td>
</tr>
</tbody>
</table>

Since $a$ and $b$ have the smallest distance (among all pairs of clusters), the algorithm merges the two points into a cluster which we denote as $S_1$. Now, there are 3 clusters left, whose mutual distances are:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>-</td>
<td>2</td>
<td>$\sqrt{13}$</td>
</tr>
<tr>
<td>c</td>
<td>-</td>
<td>-</td>
<td>$\sqrt{5}$</td>
</tr>
</tbody>
</table>

Hence, the algorithm merges $S_1$ with $c$ into a cluster which we denote as $S_2$. Now that there are only two clusters left (i.e., $S_2$ and $d$), the last merge is trivial. The following dendrogram illustrates the above process.
Max. Repeating the above algorithm with respect to max results in the following dendrogram:

\[ a \quad b \quad c \quad d \]

Problem 3. Suppose that we use \( d_{\min} \) to define the similarity of two clusters \( C_1, C_2 \). Give an algorithm to compute the dendrogram on \( n \) points in \( O(n^2 \log n) \) time.

Answer. Our algorithm maintains a BST \( T \) at any moment that stores the distances of all pairs of the current clusters.

At the beginning, each object forms a cluster by itself. Hence, \( T \) contains \( \binom{n}{2} \) cluster-pair distances.

Consider, in general, that the current clusters are \( C_1, C_2, \ldots, C_k \). We remove the smallest cluster-pair distance from \( T \). Suppose that this is the distance between \( C_i \) and \( C_j \). Then:

- We merge \( C_i \) and \( C_j \) into a new cluster \( C_{\text{new}} \).
- Delete from \( T \) the distance between \( C_i \) and every other cluster. Do the same for \( C_j \).
- Insert into \( T \) the distance between \( C_{\text{new}} \) and every other existing cluster \( C \) (i.e., \( C_1, \ldots, C_k \) except \( C_i, C_j \)).

To implement the above, the key is to compute \( d(C_{\text{new}}, C) \), namely, the distance between \( C_{\text{new}} \) and \( C \). We achieve the purpose as follows:

\[
d_{\min}(C_{\text{new}}, C) = \min\{d_{\min}(C_i, C), d_{\min}(C_j, C)\}
\]

In summary, when there are \( k \geq 2 \) clusters left, the next merge requires:

- Removing the minimum distance from \( T \)
- Deleting \( O(k) \) distances into \( T \)
- Inserting \( O(k) \) distances into \( T \).

The total time for the above operations is \( O(k \log k^2) = O(k \log k) \) (notice that \( T \) stores \( O(k^2) \) distances).

Therefore, the total running time of our algorithm is

\[
\sum_{k=2}^{n} O(k \log k) = O(n^2 \log n).
\]
**Problem 4.** Suppose that we use $d_{\text{mean}}$ to define the similarity of two clusters $C_1, C_2$. As discussed in the lecture, $d_{\text{mean}}(C_1, C_2) = \frac{1}{|C_1||C_2|} \sum_{(p_1, p_2) \in C_1 \times C_2} \text{dist}(p_1, p_2)$. Give an algorithm to compute the dendrogram on $n$ points in $O(n^2 \log n)$ time.

**Answer.** The algorithm is precisely the same as the one in Problem 3, but with one change. Recall that the key to ensure $O(n^2 \log n)$ time is to compute $d(C_{\text{new}}, C)$ in constant time from $d(C_i, C)$ and $d(C_j, C)$ when we merge together $C_i$ and $C_j$ into $C_{\text{new}}$. When $d = d_{\text{mean}}$, we can do so as follows:

$$d_{\text{mean}}(C_{\text{new}}, C) = \frac{|C_i| \cdot d_{\text{mean}}(C_i, C) + |C_j| \cdot d_{\text{mean}}(C_j, C)}{|C_i| + |C_j|}.$$ 

**Problem 5.** Consider the set $P$ of points below:

Set $\epsilon = 1$ and $\text{minpts} = 3$. Show the clusters output by DBSCAN, assuming that the distance metric is Euclidean distance.

**Answer.** First, identify the core and non-core points, shown below in black and white, respectively.

Then, the algorithm temporarily ignores the non-core points, and draws an edge between each pair of core points that are within distance $r = 1$. This creates a graph:
It proceeds by computing the connected components of the graph. In the above graph, there are 3 connected components: \( C_1 = \{a, b, c, d, e, f\} \), \( C_2 = \{g\} \), and \( C_3 = \{h, i, j\} \).

\( C_1 \), \( C_2 \) and \( C_3 \) form a cluster, respectively. In the final step, the algorithm assigns each non-core point \( z \) to each cluster that contains a core point whose neighborhood covers \( z \). Consider, for example, point \( m \). It is added to \( P_1 \) because \( m \) is in the neighborhood of \( f \). After assigning all the non-core points, we get \( \{a, b, c, d, e, f, k, m, o\} \), \( \{g, n, l\} \), \( \{h, i, j, p, q, r, s\} \) as the final clusters. Note that point \( t \) is regarded as noise.

**Problem 6.** Given a pair of parameters \( \epsilon \) and \( \text{minpts} \), describe an algorithm to compute the DBSCAN clusters in \( O(n^2) \) time, assuming that the distance metric is Euclidean distance, and that the dimensionality of the data space is a constant.

**Answer.** First, compute the distance graph. Then, discard all the edges whose weights are more than \( \epsilon \). All these can be done in \( O(n^2) \) time. Let \( G \) be the graph obtained at this moment.

For each vertex, get its degree in \( G \). It is a core point if its degree is at least \( \text{minpts} - 1 \). Otherwise, it is a non-core point. Remove the non-core points from \( G \) and their edges. Let \( G' \) be the graph obtained at this moment. All these can be done in \( O(n^2) \) time.

Now, compute the connected components of \( G' \), which takes \( O(n^2) \) time. Treat each connected component as a cluster.

For every non-core point \( u \), look at its neighbors in \( G \). If \( u \) has a core-point neighbor \( v \), add \( u \) to the cluster of \( v \). Doing so for all the \( u \) takes \( O(n^2) \) time in total.