On page 451, the proof of Theorem 17.1.1 presents an algorithm for building an ultrametric tree. The algorithm is correct, however Theorem 17.1.3 states that the algorithm can be implemented in $O(n^2)$ time. In fact, I don’t see how to do that, although $O(n^2 \log n)$ is an easy bound for the algorithm. Rather than continuing to try to find a clever implementation of that algorithm, here is another combinatorial algorithm that I claim is correct and that does run in $O(n^2)$ time. The algorithm is described in terms of a graph $G$, based on matrix $D$, but it can be implemented without an explicit graph.

Let each row $i$ of matrix $D$ be represented by a node $i$ in $G$, and each edge $(i, j)$ be given the value $D(i, j)$. In $O(n^2)$ time, the algorithm will find a very particular path in graph $G$:

Set $N$ equal to all the indices $1$ through $n$;
Set $L$ to the empty path;
Set $i$ to any node.
repeat $n − 1$ times:
begin
remove $i$ from $N$;
find an index $j$ in $N$ such that $D(i, j) \leq D(i, k)$ for any $k$ in $N$.
place edge $(i, j)$ in the path $L$;
set $i$ to $j$;
end;

What this produces is a path $L$ of exactly $n$ edges, and the algorithm can be implemented in $O(n^2)$ time. It turns out that $L$ is a minimum spanning tree of $G$, but that fact is not needed.

We will now use $L$ to create the ultrametric tree recursively.

Concentrate on an edge $(p, q)$ in the path $L$ with the largest edge weight of all edges in $L$, and let $P$ be the set of nodes at or to the left of $p$ in $L$, and let $Q$ be the set of nodes at or to the right of $q$ in $L$. The fact that $D$ is an ultrametric matrix implies that for any pair of nodes $(i, j)$ where $i$ is in $P$ and $j$ is in $Q$, $D(i, j) = D(p, q)$. One way to prove this is by induction on the number of edges between $i$ and $j$ in $L$ (applying the ultrametric condition that the in any triangle, the max of the three edge weights is not unique).

What this means is that in the ultrametric tree we are building (and in any ultrametric tree for $D$), any pair of leaves $(i, j)$ where $i$ is in $P$ and $j$ is in $Q$ must have their least common ancestor at the root of the ultrametric tree, and that root must be labelled $D(p, q)$.
If there are $k > 1$ ties for the global max edge weight in $L$, then removing those $k$ edges creates $k + 1$ subpaths of nodes, and applying the above argument, any two nodes $i$ and $j$ which are in different subpaths must have their least common ancestor at the root of the tree, which again must be labeled $D(p, q)$. Hence, any ultrametric tree $T$ for $D$ must have exactly $k + 1$ edges out of $D$, and the leaf set below any such edge must be exactly the (distinct) set of nodes in one of the $k + 1$ subpaths.

No matter what $k$ is, removing the $k$ max weight edges in $L$, and partitioning $N$, takes only $O(n)$ time.

To continue the description of the algorithm, we assume for convenience that $k = 1$. Let LP and LQ denote the two subpaths created by removing the max weight edge in $L$. Now we want to find an ultrametric tree for set $P$ and one for set $Q$; these two ultrametric trees will then be attached to the root to creat the full ultrametric tree for $D$. But note that we already have the needed paths LP and LQ that would be created if we were to recursively apply the above method (clearly LP could result if we applied the path building algorithm to $P$ alone, and similarly for LQ and $Q$). So we only need to find the max weight edge(s) in LP and the max weight edge(s) in LQ. Those two edges can be found in $O(n)$ total time. Again, because the nodes were partitioned in the first step, this time bound holds even for $k > 1$.

Continuing, we build the ultrametric tree in $O(n^2)$ total time.

Note that at each step of the algorithm, the node partitions that are created, and the associated edges that are put into $T$, are forced. Hence if $D$ is an ultrametric matrix, the ultrametric tree $T$ for $D$ is unique.