1. **Exercise 21.3-1 (p. 509)**

   Do Exercise 21.2-2 using a disjoint-set forest with the weighted union and path compression.

   Note Exercise 21.2-2 contains the following operations and it asks you to show the data structure that results and answers returned by the FIND-SET operations. In Exercise 21.3-1, You should show them when a disjoint-set forest with the weighted union and path compression is used.

   ```
   for i ← 1 to 16
   do MAKE-SET(x_i)
   for i ← 1 to 15 by 2
   do UNION(x_i, x_i+1)
   for i ← 1 to 13 by 4
   do UNION(x_i, x_i+2)
   UNION(x_1, x_5)
   UNION(x_11, x_13)
   UNION(x_1, x_10)
   FIND-SET(x_2)
   FIND-SET(x_9)
   ```

   **Exercise 23.2-2 (p. 573)**

   Suppose that the graph \( G = (V, E) \) is represented as an adjacency matrix. Give a simple implementation of Prim’s algorithm for this case that runs in \( O(V^2) \) time.

   **Exercise 23.2-8 (p. 574)**

   Professor Toole proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph \( G = (V, E) \), partition the set \( V \) of vertices into two sets \( V_1 \) and \( V_2 \) such that \( |V_1| \) and \( |V_2| \) defer by at most 1. Let \( E_1 \) be the set of edges that are incident only on vertices in \( V_1 \), and let \( E_2 \) be the set of edges that are incident only on vertices in \( V_2 \). Recursively solve a minimum-spanning-tree problem on each of the two subgraphs \( G_1 = (V_1, E_1) \) and \( G_2 = (V_2, E_2) \). Finally select the minimum-weight edge in \( E \) that crosses the cut \( (V_1, V_2) \), and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.

   Either argue that the algorithm correctly computes a minimum spanning tree of \( G \), or provide an example for which the algorithm fails.

   **Problem 23-3 (p. 577)** A bottleneck spanning tree \( T \) of an undirected graph \( G \) is a spanning tree of \( G \) whose largest edge weight is minimum over all spanning trees of \( G \). We say that the value of the bottleneck spanning tree is the weight of the maximum-weight edge in \( T \).

   (a) Argue that a minimum spanning tree is a bottleneck spanning tree.
Part (a) shows that finding a bottleneck spanning tree is no harder than finding a minimum spanning tree. In the remaining parts, we will show that one can be found in linear time.

(b) Give a linear-time algorithm that given a graph $G$ and an integer $b$, determines whether the value of the bottleneck spanning tree is at most $b$.

(c) Use your algorithm for part (b) as a subroutine in a linear-time algorithm for the bottleneck-spanning-tree problem. (Hint: You may want to use a subroutine that contracts sets of edges, as in the MST-REDUCE procedure described in Problem 23-2).