1. Assume you use breadth-first search to actually find your way out of a maze (consisting of rooms and hallways, which represent the vertices and edges, respectively). One room is the start room, and one room is the exit room. We want to count how many hallways you need to walk down. If you walk down the same hallway twice that counts as two. Each direction counts once, so back-and-forth across a hallway counts as two. The *first time* you visit a room must be in the order of breadth-first search. Other than that you can (and must) be clever.
   (a) Assume the maze is one long path of n rooms where you start on one end and the exit is at the other end. How many hallways do you walk down? Justify.
   (b) Assume the maze consists of a start room with $n-1$ other rooms directly connected to it. (In other words it is a star graph.) Assume the last room you visit is the exit room. How many hallways do you walk down? Justify.
   (c) Assume the maze is a complete binary tree with $n = 2^k - 1$ rooms. Assume that the root is the start room. To keep things simple, assume the root is also the exit room but only *after* you have visited all of the other rooms. How many hallways do you walk down? Justify.

2. Give a linear time, depth-first-search algorithm to find the size of the largest connected component in a graph, where size is measured by the number of edges in the component. (This should very similar to the algorithm covered in class.)