Homework 13
CS 3385

1. Suppose that the `connectedComponents` algorithms (given on page 563) is run on the undirected graph $G = (V, E)$ with vertices $V = a, b, c, d, e, f, g, h, i, j, k$ and the edges E are processed in the order $(d, i), (f, k), (g, i), (b, g), (a, h), (i, j), (d, k), (b, j), (d, f), (g, j), (a, e)$. List the vertices in each connected component after each iteration of lines 3-5, following the format of figure 21.1(b). Use the weighted-union heuristic and give the nodes in the actual order they would appear in the list. In case of a tie, append the set on the right to the set on the left, e.g., if a and i both have only one element in the set, (a, i) would result in a, i and (i, a) would result in i, a.

2. Consider an undirected graph $G = (V, E)$ with k connected components. Give your answers to the following questions in terms of the number of vertices $|V|$, the number of edges $|E|$ and k. During execution of `connectedComponents`,

   (a) how many times is `findSet` called?
   (b) how many times is `union` called?

3. Consider an image I where you access the intensity value of each pixel of the image with $I[x][y]$. Suppose the image is grayscale, so $I[x][y]$ returns an integer. Given a pixel p, you can use $I[p]$ as a shortcut. We wish to find all connected components where a connected component is a grouping of pixels that are connected and have identical intensity values. Two pixels are connected if they share an edge. They are not connected if they share only a corner.

   Give an algorithm for finding the connected components using disjoint sets. You'll want to start by creating and initializing a 2-dimensional array of representative nodes – one node for each pixel. You can assume you have `makeSet()` (which returns a representative node), `union()` and `find()`. You may not necessarily use all three. Describe your algorithm in pseudocode.

4. Consider the following program.

```
for i = 1 to 6
  makeSet(x_i)
for i = 1 to 6 by 2
  union(x_i, x_{i+1})
union(x_4, x_5)
union(x_1, x_4)
```

   (a) Following the linked-list representation of figure 21.2, show the data structure resulting from running the above program. Use the weighted-union heuristic. In case of a tie, use the same tie-breaker strategy as is given in #1.
   (b) Following the disjoint-set forest representation of figure 21.4, show the data structure resulting from running the above program. Use union by rank and path compression. See the algorithms on page 571. In case of a tie, make the left-hand node a child of the right-hand node. For example, the first union, which is `union(1, 2)`, would result in `1 -> 2`.

5. Write a nonrecursive version of `findSet()` with path compression. *Hint:* The simplest way is to use two loops and to not use a stack.