Final answers

1.
   a. No, don't need to sort to find min
   b. Any explanation -- for example, you can put all the elements into an array and go thru keeping track of the smallest so far
   c. No you cannot conclude it because of part B, there is a solution which is lower bound N
2.
   a. A – last seen A
      B – A
      C – D
      D – A
      E – B
      F – G
      G – E
   b. found wrong path – should have included two of them
      i. A, B, E should be A, D, C, F, E
      ii. A, B,E, G, F should be A, D, C, F
      iii. A, B, E, G should be A, D, C,F, E,G
   c. Remove C->F

3. This is the fibinoacci dynamic programming we did.

```
Int countWays(int n)
        Int[] steps = new int[n]
        For( int I = 0 ; I < n ; I++)
           If(I ==0 || I == 1)
              Steps[i] = i+ 1;
          Else
              Steps[i] = stes[i-1] + steps[i-2]
         Return steps[n-1]
```

I expected a dynamic programming solution – not recursive.

4. Yes it does because it always picks smallest weighted edge until it creates an MST

Solution 1:

There are several possibilities. Each is based on the observation that a connected component, having k vertices, must have at most $k-1$ edges to be acyclic (and at least $k-1$ to be connected). Therefore, the simplest algorithm is to run the connected components algorithm given in class, return the number of connected components, and calculate the number of edges that must be removed from there. If there are n vertices in the graph and if there are m connected components, then there must be at most $n-m$ edges remaining in the graph to make it acyclic. If there are e edges overall (easily counted) then $e-n+m$ vertices must be removed.

Alternatively, consider a modification of the connected components algorithm to count edges and vertices in each connected component. If these counts are e and v, respectively, then $e-v+1$ is the number of edges that must be removed from this component. The algorithm will then sum this number over all connected components.

Solution 2:

To identify the edges to be removed, run a DFS based algorithm to number the vertices in the order in which they are visited during DFS. Use this numbering to detect back edges (as in the algorithm for articulation points) and keep count of them. The total number of back edges is the minimum number of edges to be removed to eliminate cycles in the graph. Removing the back edges will eliminate the cycles in the graph. (Note: There must be an outer loop for the basic DFS algorithm to ensure every vertex, and hence every connected component, is visited.) The algorithm takes $O(|V|+|E|)$ since each vertex is considered once, and the adjacency list of each vertex is scanned at most once.

5.
6. A. Most everyone got credit on this -- values must be sorted and show each stage

B.

```
A:
int l=0, r=0; // combine sorted arrays
for (int i=0; i<values.length; i++) {
      if (r>=right.length || (l<left.length && left[l]<right[r])) {
            values[i]=left[l];
            l++;
      } else {
            values[i]=right[r];
            r++;
```

```
      }
}
```

c) Answer the following three questions: What is the complexity of dividing and merging an array of length *n*? What is the complexity of dividing arrays in Mergsort? What is the overall complexity of Megesort? Give your answers in big O notation. (5 points)

A: Dividing and merging arrays: O(n) time (two passes through array) (2 points)
*Dividing arrays: One division: O(n) + 2*O(n/2), Second division: O(n) + 2*O(n/2) + 4*O(n/4) ... O(log n) (2 points)*
Overall complexity: O(n log n) (1 point)

7. I wanted a discussion on what is P and NP, how it would provide a faster solution if can be proven, or help classify problems otherwise, it shows how to classify cs algorithms
8. linked list – merge sort – more memory efficient
array – quicksort – in place, no extra space to sort
extra credit: anna shaverdian