Last name _____

First name _____

LARSON—MATH 356–SAGE WORKSHEET 09
Spanning Trees!

Reminders

1. Read ahead in our textbook. Up next is Prufer codes of trees and Cayley's Theorem.

Coding Algorithms

1. Log in to your Sage/CoCalc account.
   (a) Start the Chrome browser.
   (b) Go to `http://cocalc.com` and sign in.
   (c) You should see an existing Project for our class. Click on that.
   (d) Click "New", call it **s09**, then click "Sage Worksheet".
   (e) For each problem number, label it in the Sage cell where the work is. So for Problem 1, the first line of the cell should be `#Problem 1`.
   (f) When you are finished with the worksheet, click "make pdf", email me the pdf (at `clarson@vcu.edu`, with a header that says **Math 356 s09 worksheet attached**).

Saving and Re-using Code

We've coded several graphs now, and have added code for functions of graph invariants and auxiliary functions and stored them in "graphs.sage". I pushed my updated version to your Handouts folder. Either copy that file to your Home directory—or add the new stuff to your own "graphs.sage" file. We'll need those functions.

2. I've updated the copy of "graphs.sage" in your Handouts folder to include what we've added in class. *Copy* the current version from Handouts to your Home directory.
3. *Load* your copy of "graphs.sage". Run: `load('graphs.sage')`.
4. Generate and display a `random_weighted_graph` with 5 vertices.

Minimum Weight Spanning Trees

How can we find a minimum weight spanning tree (MST) of a connected weighted graph? We discussed and implemented Kruskal's algorithm. Today we'll implement the Naive algorithm and compare running times.

5. Write a function `naive_MST(g)` that inputs a connected weighted graph g and returns a minimum weight spanning tree of that graph by (1) considering every subset (combination) of $\epsilon - 1$ edges, testing if those edges form a tree, and then checking if the sum of those edge-weights are better than a current "best".

6. It would also be useful to have a function `graph_weight(g)` that inputs a weighted graph g and outputs the sum of the edge weights of the graph. We can use it in comparing weight-sums of spanning trees. Test it!

7. Now let's compare our Kruskal and Naive algorithms to see that their outputs agree. Run some tests!

8. Now let's compare the speed (runtimes) of these algorithms. One simple way is to use Sage's `timeit` command. Evaluate `timeit?` to see if the documentation for this function has any useful comments, parameters or examples.

9. Use `timeit` to run some comparisons of the runtimes of the Kruskal and Naive algorithms for random weighted graphs of various orders.

10. What is the *average* (expected) weight of a minimum weight spanning tree of a complete K_ν with various orders nu. Give the edges random weights in [10].

Recursive Functions & a Tree theorem

I claimed that a graph G is a tree if and only if it has a degree-1 (leaf, pendant) vertex v and $G - v$ is a tree. We can use this idea to write a *recursive* function for testing whether a graph is a tree: see if it has a leaf, peel it off, and repeat for the remaining graph.

A **recursive** function is a function that calls itself. It must always have a *base case* so that the recursion eventually stops.

11. Here is an example of a recursive definition of the *factorial* function. The base case here is the case where the input is 0 or 1.

```
def factorial(n):
    if n==0 or n==1:
        return 1
    else:
        return n*factorial(n-1)
```

Now try `factorial(0)`, `factorial(1)`, `factorial(2)`, `factorial(3)`, and `factorial(10)`.

12. The Fibonacci numbers are: 1,1,2,3,5,8,13,21..., where the next number is the sum of the previous two numbers. Write a recursive function function that computes the n^{th} Fibonacci number.

13. Write a function `is_tree_recursive` to test whether a graph is a tree by seeing if it has a leaf, peeling it off, and repeating for the remaining graph. You'll need a base case. What will it be?