Announcements

- Assignment 3 is posted
- Read the entire handout before starting
- Notice the weights in the marking section
- Write your test cases as you write your a3_functions.py functions... not after!
- Design your functions before implementing them.

Sorting

The problem

- Take a list of items and put them into order, according to some key. Examples:
  - sort ints by value
  - sort strings by length
  - sort students by student number
- We might sort into non-ascending or non-descending order.
- Why don't I just say descending or ascending order?

Why is sorting important?

- Searching can be significantly faster if the list is sorted.
- And sometimes we want a sorted list for other reasons.
  Example: A user may want to see book recommendations in order by their star-rating

Sorting techniques

- In real life, how would you sort:
  - Playing cards in your hand?
  - A stack of midterms
- There are many sorting techniques!
- We'll compare several.

Sorting demos

- Terrific website compares various algos: http://www.sorting-algorithms.com/

Comparing growth rates

Time required to sort random lists of various sizes (in a very small experiment):

Algorithm	size 1,000	size 2,000	size 4,000	size 8,000
bubble sort	0.2	0.9	3.6	14.6
insertion sort	0.13	0.5	2.2	8.4
selection sort	0.1	0.4	1.6	6.5
merge sort	0.01	0.02	0.05	0.11
list.sort	0.0005	0.001	0.002	0.005

What to remember about sorting

- The algorithms (enough to act them out)
- The invariant for each algorithm
- The growth rate of each of the algorithms
- Do not try to memorize the code!

Implications beyond sorting

- There can be massive differences in run-time between different algorithms for the same problem.
- Which algorithm is fastest may depend on the particular values being used.
  Eg, a random list vs a nearly sorted list.
- Some algorithms would take so long (on problem sizes you might want to solve) as to be useless.
- Solution: Try to find a faster algo?   But...

An even bigger idea

- There are some problems for which no one has ever found a "fast" algorithm.
- Eg: travelling salesperson problem (TSP).
- If you are solving a problem that amounts to the TSP, it would be good to know!
- Then what do you do? There are techniques for finding an approximate solution.
- Very smart people have been trying to prove that **there is no fast solution** for the TSP **or a whole category of problems.**
- This is all part of what makes CS a science.