Week 7 Lab: Implementing Radix Sort

[If you did not attend lecture on Wednesday, you have some catching up to do]

The goal of this lab is simple: let's see if you and a partner can write a complete Radix sort program in 50 minutes!

Guidelines:

Find a partner to work with.

Program behavior: when run, your program simply reads a sequence of non-negative integers from standard input just like your quick-sort implementation. It produces the sorted sequence to standard output.

After the sorted output, your program should print a short report:

```
N:         number of items sorted
MAX:       largest value among the input
NPASSES:   the number of passes the algorithm made
```

You are not given any "scaffolding" -- you will put everything together yourself! Be independent! Figure things out!

However, you may leverage things (code) we have used during the semester. It is up to you to decide what those things should be and how to put everything together.

Radix: your program may simply use a radix of 10. But feel free to implement strategy that selects the radix to optimize runtime.

Keep things as simple as you can: Don't shoot for a super-tuned implementation -- you only have 50 minutes. For example, if your implementation does more mallocs and frees than necessary, don't worry about it right now.

How to start? Discuss the basic design decisions:

- "How will we read the input and where should we initially store it?"
- "What data structures should we use?"
- "What code from previous programming exercises might be useful?"
- "How should we organize the code -- multiple files? How will we compile and link?"
- "Can we identify sub-problems / functions what we can work on independently and then integrate?"
- SUMMARY: spend 10 minutes with your partner devising a plan! You can always modify your plan, but it is still usually best to start with one!

Receiving Credit:

To receive credit for this lab, you must flag down a TA and show them what you have done. You might not arrive at a working program, but you need to be able to show substantial progress!

Above-and-beyond:

It would be interesting to have a basic, correct, but not super-optimized implementation first (ie, from this lab) and then write an optimized version (e.g., by being clever about division and modulus; avoiding unnecessary malloc/free, etc.). Then you could do some experiments to see how much speedup you got!