Dan Caruso – Week 6 Lab Analysis for Array and Linked List Implementations:

When both implementations have been finished provide a written analysis of the two approaches. Compare and contrast both designs including pros and cons of each. Where does the array implementation perform well? What are the characteristics of arrays that contribute to this performance? Where does the linked list perform well? What are the characteristics of linked lists that contribute to this performance? Are there any tradeoffs? Is there anything that needs to be sacrificed to achieve the good performance? Can you quantify?

After spending quite a bit of time with each implementation, I must agree with a main point driven home in the lecture: dynamic array implementation is a lot more logical to read, follow, and construct than linked lists, though it is necessary to consider this bias from the standpoint of having constructed arrays and lists for a few semesters now. For example: it was a lot easier to follow my debugger when testing something that dealt with an index further along into the list since it was easier to kick realistic values out as I followed; with a linked list I had to get used to paying attention to following memory addresses to make sure that things were flowing smoothly.

I found that I had an easier time with arrays performing well when having to append a list to the end of the current list, mainly due to the need for getting pointers correct (I had a harder time getting that squared away with a linked list). It could just be how I wrote things, but testing my array functions seemed to process a little faster than their linked list counter parts, especially when debugging. There seemed to be less shuffling around of data with a few things requiring fewer steps (example: remove duplicates), though generally speaking from what I can gather most functions for both approaches operate on an O(n) degree of functionality on average case scenarios. Your mileage of n may vary depending on the task and how much iteration is necessary, but neither approach was bogged down with O(n^2) level algorithms.

Linked lists had the advantage of being able to resize and insert much easier once I got the hang of it: once you traverse to the correct spot, albeit snaking your way through pointers and references as opposed to just paying attention to what index you're on, a re-mapping of pointers was all it took, no shuffling of data items necessary. Arrays were more annoying given the shifting of data that had to occur when inserting or removing something, with the added annoyance from having to resize everything. I actually went back through and revised a bunch of these types of functions in my array implementation AFTER having completed my linked list implementation after realizing this.

Overall it was a give and take, as I kept solving things in one implementation it would be an eye opener for the other; I still may prefer arrays but I can appreciate the functionality of linked lists. Admittedly, I wasn't able to connect the dots at the very end with the dorm application: I was able to successfully read in each file, build my student list, and implement a decent randomization of dorm assignments through "shuffling," but had trouble approaching an unordered map technique to put it all together with a linked list at the end.