**Lesson 9 – Array Sorting**

* Variety of ways to compare algorithms
  + **Algorithm Complexity** = How confusing the code is to write
  + **Algorithm Structure** = Basic method we will use to sort the array
  + **Computational Complexity** = How hard it is for the computer to sort the data
  + **Memory Usage** = How much extra memory is required for the algorithm
  + **Array Stability** = How likely is the array to stay sorted
* **Algorithm Complexity** 
  + Comparison of the overall complexity of the code required to write the algorithm
    - SelectionSort, BubbleSort, InsertionSort are simple
* **Algorithm Structure**
  + Comparison of the general style or type of algorithm
    - Can be:
      * Swap (Switching paired items)
      * Merge (divides array into sorted and unsorted parts)
      * Tree (places data in a binary tree)
      * Other (generally more complex)
    - Bubble Sort
      * Swap Based (Swapping individual elements)
    - Selection Sort
      * Swap Based (Swapping individual elements)
    - Insert Sort
      * Merge Based (merging unsorted to sorted side)
* **Computational Complexity** 
  + Comparison is based on the processing time of the algorithm
    - Based on the number of comparisons made since it can be difficult to compute the number of swaps made
    - Processing time can change because of the complexity/amount of disarray of the array to be sorted
      * Results in analyses of best, worst, and average case scenarios
  + Generally, programmers want to know the efficiency of their programs when there is a large amount of data (n)
    - Evaluated using O Notation
      * A mathematical language for evaluating the running-time of algorithms
      * Gives an approximate rate of growth for an algorithm (e.g. linear vs quadratic)
      * We typically evaluate the worst-case scenario
      * Examples:
        + Bubble Sort with an array of 10 elements

45 total comparisons

In general, there are N-1 comparisons each time

Formula: N \* (N-1)/2

Which has a growth rate of O(N^2) comparisons

* + - * + Selection Sort with an array of 10 elements has the same computational complexity as Bubble Sort but it sorts much quicker (10 swaps for 10 elements)
        + Insert Sort

This algorithm is twice as fast as Bubble Sort since on average only half the number of comparisons is performed for each pass before the term is found

* **Memory Usage**
  + Used to compare the amount of memory required to sort the array (not including the size of the actual array)
    - All 3 algorithms require very little extra memory
      * Only 1 variable is used for temporary storage during data swapping
* **Stability**
  + Used to see if data reminds in the same order when sorted for multiple attributes
    - Example:
      * If you sort a list of people by first name that have already been arranged alphabetically by last name, do the names with the same first name remain sorted alphabetically by last name?
        + If yes, it is stable if not it is not stable
    - All 3 algorithms are stable