**Lab05 Results Discussion**

**Jason Witherell**

**Explanation of project**: In this lab, I repeatedly created ArrayLists of increasing sizes and performed a few operations on that data, measuring the time of each in milliseconds (using the C++ chrono library). The results were then graphed and compared.

The first test was performed on data of size 1000. Each subsequent test size was increased by 5000 up to 400,000 (396,000 ended up being the highest)

The operations performed on each ArrayList were:

* **Create** the ArrayList of floats (each value was 0.5 higher than the index: 0.5, 1.5, 2.5, etc.), called AL.
* Use the Fisher Yates **shuffle** algorithm from Lab5 to scramble the ArrayList.
* Make a **copy** of the ArrayList called AL\_copy.
* **Save** away 10% the values (the first 10%) of AL in another ArrayList called find\_values
* **Quicksort** AL in ascending order.
* **BubbleSort** AL\_copy (both AL and AL\_copy should have same values)
* Find all of the find\_values in AL using **BinarySearch**
* Find all of the find\_values in AL using **LinearSearch** (find method from Lab2)

**Analysis:**

1. The time to create, copy, saving away the 1000 values, and binary-search were very fast so I measured in nanoseconds. Most of them scaled with ArrayList size except for the reservation (because it was always 1000 elements). The times were as follows:
2. BinarySearch and LinearSearch were the next most expensive items. I measured both in nanoseconds and the time to find the 1000 reserved values. When shown together as in this figure, BinarySearch looks flat:

You can see a difference in grown when BinarySearch is shown by itself:

1. QuickSort was the next most expensive item and grew as follows (this is the cost to sort the shuffled list into ascending order):
2. BubbleSort was where the vast majority of time was spent in this program (99% of the overall time for everything was spent in this algorithm). In fact, I had to stop performing it early or it would’ve taken many more hours! Here is a graph of the time taken (in ms) as a function of the ArrayList size, shown on a normal y-axis scale and a log-10 scale.