ETEC2101	Lab4: big-O introduction and a few algorithms	Points: 100

1. Get a working copy of Lab3 to build off of.
2. (**25 points**) Make a few templated *functions* (you can make more than these – there are just the ones the user should have access to) in a new **array_list_utility.h** file
    a. **quicksort**: take an ArrayList reference, an enum (class) value indicating whether the user wants to sort in ASCENDING (small => big) or DESCENDING (big => small) and sorts the list "in-place" using the quicksort algorithm as discussed in class
        i. The method should return the number of swaps that were performed.
        ii. You will likely need to create a few "helper" functions to assist in this operation
    b. **binary_search**. Should take an ArrayList const-reference, a sort-order enum value, a search value and an optional pointer to a long-int [more on that below]. The method should return the index of (one of) the occurrence(s) of the value in the arraylist or -1 if it's not found. If the user passes a valid pointer to you, use it to record the number of comparisons that were made during the binary search operation.
    c. **bubble_sort**. Should take similar arguments and have a similar return value to quicksort, but should use the bubble-sort algorithm to sort.
    d. **shuffle**: Use the Fisher-Yates algorithm to shuffle. Should re-arrange a passed ArrayList (reference) to be in a randomized order. Use the c++ <random> module rather than rand (just so you've seen it).
3. (**15 points**) Test the above methods using new gtest tests (I put these in a new array_list_utility_tests.cpp)
4. (**40 points**) Compare running-times
    a. Do this by creating two alternative main programs that you can toggle between with a macro
    b. Perform the following automated tests on a randomly-generated ArrayList of floats of sizes 1000 to 250,000 (by steps of 5000)[1]. Make sure this is automated (i.e. don't run the 1000-size test, record the result, then manually re-run the 2000-size test, etc.)
    c. Measure the time taken (in milliseconds, using the <chrono> library) of each of these operations:
        i. Shuffle the ArrayList
        ii. Making a copy of an ArrayList (using our copy-constructor)
        iii. Record the starting value (we'll search for this in steps vi and vii)
        iv. Sort the *original* using mergesort
        v. Sort the *copy* using bubble-sort
        vi. Find an element (that we found in step iv) using binary search
        vii. Find an element (that we found in step iv) using linear search (i.e. the find method from Lab1)
    d. For items 4biv – 4bvii, also record the number of operations (swaps for sorting, comparisons for searching). Since linear search starts from the beginning until it finds the element, the index of the found item is the same as the number of comparisons. For linear and binary search, calculate the average of finding some number of elements (half, for example), so we don't just get lucky and find the element quickly, skewing the results.
    e. Output this data to a text file.
    f. (+5 points) for getting to 500,000 array-list sizes instead of 250,000.
5. (**20 points**) Graph the results of your tests[2] and collect the graphs in a document. You can just show the graphs if you sufficient label them or you can include descriptive text describing what the graph represents if not. If a graph is hard to read (as BubbleSort might be), you might want to use a logarithmic axis scale.
6. (**up to +20 points**) Try implementing another sorting algorithm or to and measure their performance. MergeSort and InsertionSort are a few to consider. I'll give +10 for each additional sort, up to a max of +20.
    a. Just for fun: https://www.youtube.com/watch?v=kPRA0W1kECg (volume warning!)

[1] This took my mid-level laptop about 30 minutes to complete this in RELEASE mode (Release is generally much faster than Debug!). The 500,000 version took over 4 hours to complete!

[2] If you're not familiar with Excel / Google Sheets / etc. I'll be happy to give you a quick tutorial (just ask)