Samantha Ho
COSC 160
September 29th, 2019
(Extension granted by Prof. Buffum)

Project 1: List Efficiency Cover Sheet

My code for this project reads in a text file of integers and stores the integers into two different list structures before performing operations on them. The goal of this project is to compare the time efficiency of the two different list structures and their respective operations. I was able to empirically record the precise elapsed times by using **chrono**, a C++ namespace used to handle date and time.

Two list structures were required for the project: **orderedList** and **MTFList.** I have produced an implementation .cpp file and a header declaration file for each type.

The methods of the orderedList struct included a merge sort implementation and a binary search implementation. I have chosen a standard array implementation for the merge sort method. Upon research, I found that a linked list implementation would actually be the most reliable implementation choice for merge sort. This is due to the fact that the time efficiency remains the same for both linked list and array implementation, at **O(nlogn)**. However, linked list is advantageous in that it saves auxiliary space. The array implementation would require additional space complexity of **O(n),** given that subarrays need to be created in order to aid in the sorting algorithm, and the additional space required would be n. However, a linked list implementation, despite being very difficult to code, would save the auxiliary space as nodes are not contiguous in memory. Their pointers would make reorganizing them very time and space efficient. Despite this, I ultimately decided to go with an array implementation instead, because I realized the need for the program to call a searching method after sorting, and searching methods with linked lists would require traversal, with a time complexity of **O(n)**. Arrays, however, support much more efficient searching methods with time complexity of **O(nlogn)**. I chose binary search because of this reason. Since we were instructed to optimize for time efficiency over space if we were to make a judgment call, I chose this implementation because the space benefits of the linked list implementation simply do not outweigh the time benefits of array implementation. Currently I estimate space complexity to be **O(n)** and time to be **O(nlogn).**

For the **MTFList** structure, a self-organizing list where the most recently searched item would move to the front of the list, I chose a linked list implementation. This was an exceptionally easy decision to make given the list did not need to be sorted afterwards. Relinking the nodes are time efficient because they are not contiguous in memory. Traversing the list to find the item would be time complexity of **O(n)** in the worst case, O(n/2) in the average case, O(1) in the best case, and the few extra steps to relink nodes and reposition pointers will not affect the worst case heavily. Space complexity is estimated to be **O(n)** as well.