Alexa Brown

Project 6

Although insertion sort logically seems to be quicker, there doesn’t seem to be a drastic difference in the amount of time it takes for a vector/array compared to a linked list, yet there are more differences as the lists get larger. Linked lists are much quicker to insert/delete an element by simply moving the pointers. However, in order to find where the element would fit into the list, you must traverse again over the entire list, taking up more time than simply inserting or deleting an element. As well, arrays are much quicker to find the element in the list by using [ ] or .at( ) to quickly get to that element location. However, in order to insert or delete an element, you must move each element after the element you wish to insert/delete, taking up more time than just simply finding the element. Therefore, the shorter the list, the faster insertion sort works on the linked lists because it takes less time to iterate through the list as well as less time to insert the element. So, the longer the list, the faster insertion sort works on arrays/vectors because it can directly go to where the element should belong and begin moving the remaining elements after it. As we see through the program and the graph, they both take relatively the same amount of time (a miniscule amount of milliseconds) when the length is smaller, yet as it gets larger, the linked list time gets longer as well. With vectors and arrays, insertion sort is more consistent with the time it will take no matter the size of the vector, yet with linked lists, the time varies more depending on the length of the list. In real life, I would lean more towards using insertion sort with arrays/vectors because you know it will be more consistent timing. As well, often in fields with lots of data and information, you want to be able to find a specific piece of information quickly, so an array or vector would be the stronger choice for this.