CS3310

Assignment 1

Bryan Greener

Due: 2017-09-28

The algorithms used in this program are all standard. I used a basic linear search for the unsorted array, a sort function from the C# Arrays library, and a textbook example of a binary search with a tiny addition to allow me to ignore duplicates.

Theoretically, the linear search should be O(n) time complexity. The stock Arrays.Sort method uses quicksort and is therefore O(nLog(n)). Finally, my implementation of a binary search is supposed to be O(log(n)) since it is a standard binary search however it seems to have a time complexity of nLog(n).

The following data comes from searches where the input string was “Bryan Greener”.

|  |  |  |
| --- | --- | --- |
| Dimension | Unsorted | Sorted |
| 5 | 249 | 157 |
| 10 | 461 | 405 |
| 100 | 459 | 708 |
| 200 | 379 | 795 |
| 300 | 330 | 1645 |
| 400 | 420 | 1247 |
| 500 | 389 | 1083 |
| 1000 | 483 | 1136 |

Clearly this data doesn’t line up at all with the theoretical time complexities of each of these algorithms. This can be explained for both the unsorted and sorted searches. For the unsorted linear search, no matter how big the array gets, the search characters will almost always be within the first 100 characters in the array since it is completely random. As for the sorted array, the time continues to increase because as the array size increases, the number of times the binary search algorithm has to halve the search area also increases linearly. The only time that we would see my implementation of these algorithms match up to their theoretical time complexities would be if the input string contains large amounts of repeated characters.