Kevin Blum

10/9/15

CSI-281-05

Prof. Hall

PA3 Report

The purpose of this experiment is to test the scalability of the array, linked list, and doubly linked list by timing the functions to add, delete and search for an element within a sorted and unsorted list of increasing size, from 10 to 10,000 elements. This tests the effectiveness of each data structures ability to perform these functions with increasing size. The array is the only data structure already given and the linked lists must be implemented using structs, classes, and functions within the classes to perform the actions necessary to create the linked lists. In order to test each data structure I simply used a combination of for loops to perform the number of tasks I wanted for the given operation and the clock\_t to time how long the total number of operations took to complete, I repeated these for loops for each increasing size of each data structure to the gather larger sample sizes.

The data results are given in milliseconds because many operations took very little time to complete, the array took nearly no time at all for all operations performed but that is likely because it has a preset size and limit and cannot scale well without manually increasing the size. The two linked lists both had very similar times when it came to each operation, they both had a correlation where under 1,000 elements took less than a millisecond but once over 1,000 the time consumed increased much more rapidly especially seen in the search where is takes from 11 milliseconds at 1,000 elements to over 300 milliseconds at 10,000 elements.

This experiment went about as much as I expected because as the size increased the time increased for the Linked Lists but not for the Array because the array has a set limit and for searching it knows each index of each value whereas the linked lists must traverse the list each time to find the correct search term. As size increases generally speaking the run time increases which makes logical sense given that the greater amount of elements, the more actions must be performed therefore taking more time in the end. When it comes to sorted versus unsorted it generally doesn’t make too big of a difference, there may be a small variance but when it came to linked lists the step up from 1,000 to 10,000 elements provided the same increase in run time give or take so, generally speaking it doesn’t matter whether or not the list is sorted. There is not necessarily a clear winner because it depends on the needs of the program, arrays definitely perform the quickest when it comes to insertion but arrays do not scale well when the size limit is unknown, so when given an unspecified size linked lists can be better.

|  |  |  |  |
| --- | --- | --- | --- |
| **Insertion (Sorted List)** | | | |
| **N** | **Array** | **Linked List** | **Doubly Linked List** |
| 10 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 100 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 1,000 | 0 milliseconds | 1 milliseconds | 0 milliseconds |
| 10,000 | 1 millisecond | 8 milliseconds | 8 milliseconds |

|  |  |  |  |
| --- | --- | --- | --- |
| **Insertion (Unsorted List)** | | | |
| **N** | **Array** | **Linked List** | **Doubly Linked List** |
| 10 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 100 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 1,000 | 0 milliseconds | 1 millisecond | 1 millisecond |
| 10,000 | 1 millisecond | 8 milliseconds | 7 milliseconds |

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Array** | **Linked List** | **Doubly Linked List** |
| 10 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 100 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 1,000 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 10,000 | 0 milliseconds | 1 millisecond | 1 milliseconds |

|  |  |  |  |
| --- | --- | --- | --- |
| **Deletion (Unsorted List)** | | | |
| **N** | **Array** | **Linked List** | **Doubly Linked List** |
| 10 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 100 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 1,000 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 10,000 | 0 milliseconds | 1 millisecond | 1 millisecond |

|  |  |  |  |
| --- | --- | --- | --- |
| **Search (Sorted List)** | | | |
| **N** | **Array** | **Linked List** | **Doubly Linked List** |
| 10 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 100 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 1,000 | 0 milliseconds | 11 milliseconds | 12 milliseconds |
| 10,000 | 0 milliseconds | 376 milliseconds | 334 milliseconds |

|  |  |  |  |
| --- | --- | --- | --- |
| **Search (Unsorted List)** | | | |
| **N** | **Array** | **Linked List** | **Doubly Linked List** |
| 10 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 100 | 0 milliseconds | 0 milliseconds | 0 milliseconds |
| 1,000 | 0 milliseconds | 2 milliseconds | 5 milliseconds |
| 10,000 | 0 milliseconds | 374 milliseconds | 364 milliseconds |

**Deletion (Sorted List)**